Využití aplikačního frameworku Juce pro výuku programování

Petr Svoboda

Bakalářská práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr Svoboda**
Osobní číslo:	**A17152**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Využití aplikačního frameworku Juce pro výuku programování**
Téma práce anglicky:	**The Use of the Juce Application Framework for Teaching Programming**

Zásady pro vypracování

1. Popište framework Juce a související technologie.
2. Zhodnoťte možnosti frameworku z hlediska využití pro výuku programování.
3. Navrhněte úkoly vhodné pro výuku programování.
4. Vytvořte zadání a vzorová řešení úloh pro studenty.
5. Popište klíčové prvky řešení.
6. Demonstrujte výsledky.

Rozsah bakalářské práce:
Rozsah příloh:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ROBINSON, Martin. Getting Started with JUCE. 1. Birmingham: Packt Publishing, 2013. ISBN 978-1783283316.
2. PIRKLE, Will. Designing Audio Effect Plugins in C++: For AAX, AU, and VST3 with DSP Theory. 2. New York: Routledge, 2019. ISBN 978-1138591899.
3. PIRKLE, Will. Designing Software Synthesizer Plug-Ins in C++: For RackAFX, VST3, and Audio Units. 1. New York: Routledge, 2014. ISBN 978-1138787070.
4. STROUSTRUP, Bjarne. A Tour of C++ (C++ In-Depth Series). 2. New York: Addison-Wesley Professional, 2018. ISBN 978-0134997834.
5. GREGOIRE, Marc. Professional C++. 4. Indianapolis: John Wiley, 2018. ISBN 978-1119421306.
6. *JUCE* [online]. 2019 [cit. 2019-11-13]. Dostupné z: https://juce.com/

Vedoucí bakalářské práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **28. listopadu 2019**
Termín odevzdání bakalářské práce: **15. května 2020**

L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Ve Zlíně dne 9. prosince 2019

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Petr Svoboda v. r.
podpis diplomanta

ABSTRAKT

Cílem práce bylo zhodnotit možnosti frameworku JUCE a navrhnout úkoly vhodné pro výuku programování. Teoretická část se zabývá seznámení čtenáře s frameworkem, popisuje použité prvky z jazyka C/C++, vysvětluje použitelnost několika tříd frameworku, seznamuje čtenáře s Audio Plugin Hostem, a popisuje jednotlivé typy aplikací, které je možné pomocí JUCE tvořit. Praktická část se zabývá vysvětlením klíčových prvků řešení navržených úkolů.

Klíčová slova: framework, JUCE, plugin, C, C++, Visual Studio, výuka programování

ABSTRACT

The aim of the work was to evaluate the possibilities of the JUCE framework and to design tasks suitable for teaching programming. The theoretical part deals with acquainting the reader with the framework, describes the elements used from C / C ++, explains the usability of several classes of the framework, describes interaction with Audio Plugin Host, and show the types of applications that can be made using JUCE. The practical part deals with the explanation of key elements of solving the proposed tasks.

Keywords: framework, JUCE, plugin, C, C++, Visual Studio, teaching programming

Tímto děkuji vedoucímu mé bakalářské práce, panu Ing. Eriku Králi Ph.D. za vedení a rady při tvorbě této práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **9**

I TEORETICKÁ ČÁST **10**

1 JUCE FRAMEWORK **11**

1.1 PROJUCER 11

1.2 DOSTUPNÉ ZDROJE 11

1.3 ZHODNOCENÍ FRAMEWORKU Z HLEDISKA VYUŽITÍ PRO VÝUKU PROGRAMOVÁNÍ 11

2 JAZYK C++ **13**

2.1 VISUAL STUDIO 13

2.1.1 Překladač gcc 13

2.2 DALŠÍ VÝVOJOVÁ PROSTŘEDÍ 13

2.2.1 Eclipse 13

2.2.2 Qt Creator 14

2.2.3 CLion 14

3 PŘEHLED POUŽITÝCH FUNKCÍ A TŘÍD **15**

3.1 Z JAZYKA C/C++ 15

3.1.1 Lambda funkce 15

3.1.2 Enum 16

3.1.3 Struktury 16

3.1.4 auto 16

3.2 Z FRAMEWORKU JUCE 17

3.2.1 FileChooser 17

3.2.2 AudioFormatManager, AudioFormatReader, AudioFormatReaderSource 17

3.2.3 AudioTransportSource 17

3.2.4 Slider 17

3.2.5 ComboBox 18

3.2.6 Button 18

3.2.7 Point 18

3.2.8 Path 18

3.2.9 Maths 19

4 PRÁCE SE ZVUKEM POMOCÍ KNIHOVY JUCE **20**

4.1 FILTR S NEKONEČNOU IMPULZNÍ ODEZVOU 20

4.2 MODUL DSP 20

4.3 RYCHLÁ FOURIEROVA TRANSFORMACE 20

5 PRÁCE S PROJUCEREM **21**

5.1 ZALOŽENÍ NOVÉHO PROJEKTU 22

5.2 ZÁKLADNÍ STRUKTURA PLUGINU V JUCE 23

5.3 PRÁCE S PLUGINY 23
5.3.1 Přidání nových pluginů 24
5.3.2 Propojování jednotlivých pluginů v Audio Plugin Hostu 26
5.4 ANIMOVANÁ APLIKACE 26
II PRAKTICKÁ ČÁST 28
6 VYPRACOVANÉ ÚKOLY 29
6.1 ÚVOD 29
6.2 ÚKOL Č. 1 – AUDIO PŘEHRÁVAČ 29
6.2.1 Stanovení zadání úkolu 29
6.2.2 Řešení 30
6.2.3 Vzhled aplikace 30
6.2.4 Otevření souboru 31
6.2.5 Úprava hlasitosti 34
6.2.6 Využití funkcí pluginu k přehrávání 34
6.2.7 Propojení Editoru a Processoru 36
6.3 ÚKOL Č. 2 – TVORBA FILTRU 38
6.3.1 Stanovení zadání úkolu 38
6.3.2 Řešení 38
6.3.3 Vzhled aplikace 39
6.3.4 Filtr 40
6.3.5 Propojení Editoru a Processoru 43
6.4 ÚKOL Č. 3 – VIZUALIZACE AUDIO STOP 43
6.4.1 Stanovení zadání úkolu 44
6.4.2 Řešení 44
6.4.3 Spojení úkolů v Audio Plugin Hostu 49
6.5 ÚKOL Č. 4 – ANIMACE V JUCE 51
6.5.1 Stanovení zadání úkolu 51
6.5.2 Řešení 52
6.5.3 Struktury 52
6.5.4 Deklarace proměnných 52
6.5.5 Konstruktor, naplnění struktur 54
6.5.6 Pomocné funkce 56
6.5.7 Funkce resized 57
6.5.8 Funkce paint 58
6.5.9 Funkce update 59
ZÁVĚR 61
SEZNAM POUŽITÉ LITERATURY 62
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 65
SEZNAM OBRÁZKŮ 66
SEZNAM PŘÍLOH 67

ÚVOD

JUCE Framework je jednou z nově rozvíjejících se knihoven pro tvorbu audio pluginů nebo mobilních a desktopových aplikací, vhodných zejména pro hudební průmysl. Nabízí vysokou škálu grafických prvků, lze v něm tvořit různé animace, využívat OpenGL nebo vytvářet pluginy a knihovny pro úpravu audia či videa. Výhodou frameworku jsou již předem připravené šablony několika typů aplikací, které jsou navrženy tak, aby programátor již po doplnění několika málo řádků viděl reálný výsledek. Dalším plusem je relativně rozsáhlá dokumentace frameworku, kde téměř každá funkce má svůj podrobný popis, či rozsáhlé DEMO projekty, ze kterých lze vycházet.

Tato práce se zabývá seznámením s tímto frameworkem a zhodnocení jeho možností pro využití ve výuce programování. Popisuje některé funkce a jejich použitelnost. Zabývá se zejména zpracováním digitálních signálů, jejich využitím a demonstrací na jednoduchých aplikacích. Snahou je předvést studentům reálné využití věcí, se kterými se mohli setkat ve výuce.

Cílem je navržení několika dílčích úloh, které by studenti měli být schopni vytvořit se znalostí programování v jazyce C/C++ a seznámením se s několika třídami, které jsou již v JUCE připraveny.

I. TEORETICKÁ ČÁST

1 JUCE FRAMEWORK

Jedná se o multiplatformní framework, vyvíjený firmou ROLI [1]. Pro osobní užití a studenty je k dostání zdarma. Lze jej využít k tvorbě mobilních a desktopových aplikací nebo pluginů zaměřených na hudbu a zvuky obecně. Velkou výhodou tohoto frameworku je možnost využití takzvaných MIDI příslušenství [2].

1.1 Projucer

K práci s JUCE frameworkem je potřebný tzv. Projucer [3]. Ten slouží k zakládání projektů, které poté můžete spustit a vyvíjet i v jiném vývojovém prostředí. Je uzpůsoben tak, aby bylo možno aplikace vyvíjet na více platformách zároveň – podporuje Xcode, Visual Studio, Code::Blocks, ale i Android, Linux Makefiles a nově i CLion. Spolu s Projucerem je k dostání mnoho DEMO projektů a tutoriálů [4], které uživatele seznámí se základními i pokročilými třídami JUCE frameworku.

1.2 Dostupné zdroje

Framework má docela rozsáhlou dokumentaci [5], také lze koupit knihu „Getting started with JUCE“ [6], která vyšla v roce 2013, jejímž autorem je Martin Robinson. Ta může sloužit jako inspirace či úvod do začátků s JUCE, každopádně za posledních 7 let se framework velmi změnil a kniha nebyla dosud aktualizována, kdežto tutoriály na stránkách frameworku se neustále aktualizují. Dalším velmi dobrým zdrojem je YouTube kanál uživatele „The Audio Programmer“ [7]. Aktuálně tam lze najít více než 65 různých tutoriálů i s komentářem a vysvětlením autora a v orientaci v základních třídách může být velmi nápomocný. Za zmínku stojí i fórum frameworku [8], kde se již spousta věcí a případných chyb řešilo a lze zde získat zpětnou vazbu i od samotných tvůrců frameworku.

1.3 Zhodnocení frameworku z hlediska využití pro výuku programování

Existuje veliké množství různých frameworků, které mají své výhody i nevýhody, JUCE je na tom také tak. Jednou z výhod, které pro studenty a výuku nabízí je ta, že jej lze za tímto účelem stáhnout zcela zdarma. Instalace Projuceru nevyžaduje administrátorská práva a výsledné aplikace jsou přeložitelné i v samotném Visual Studiu bez potřeby mít Projucer nainstalovaný. Šablony pro jednotlivé aplikace jsou výborně zpracovány a je opravdu jednoduché při dopsání několika málo řádků kódu vidět reálný výsledek. Audio Plugin Host, který slouží k propojování jednotlivých pluginů, může být při tvorbě aplikací skvělým

pomocníkem. I tak má JUCE i své nevýhody. Je frameworkem velice specificky zaměřeným a v tomto ohledu i docela rozsáhlý, každopádně lze najít rozšířenější i rozsáhlejší frameworky (př. Qt), které jsou schopny pojmout větší škálu aplikací. Pokud se tedy bavíme o výuku programování, je nutno, před jeho nasazením ve výuce, zvážit, jestli je vhodnější než kterýkoliv z těchto frameworků.

2 JAZYK C++

Jazyk C++ vychází z jazyka C. S jeho vývojem začal v roce 1979 pro potřeby distribuce jádra UNIXu napříč multiprocesory a lokálními sítěmi dánský programátor Bjarne Stroustrup. Jméno jazyka vymyslel Stroustrupův kolega Rick Mascitti v létě roku 1983 jako náhradu za do té doby používaný název „C with Classes“. Cílem při tvorbě tohoto jazyka byla touha vyhnout se programování v assembleru a ulehčit psaní dobrých programů. Jazyk C, ze kterého C++ vychází, lze označit jako jeho podmnožinu. C++ se momentálně vyskytuje opravdu všude. Je využíván i firmami jako je Amazon, Facebook nebo Google.[9]

Od roku 1998 je jazyk standardizován a o jeho vývoj se stará ISO/IEC JTC1/SC22/WG21 [10].

2.1 Visual Studio

Jedná se o vývojové prostředí od společnosti Microsoft. V dnešní době se využívá pro tvorbu vysoké škály aplikací a podporuje více jazyků – např. C/C++, C#, Python apod. Jedná se o propracované IDE, které obsahuje editor kódu, debugger, různé vizuální designery a mnoho dalších nástrojů. Jedná se o velmi rozšířený nástroj, který je neustále vyvíjen. Aktuální verze je Visual Studio 2019.

Za zmínku stojí určitě i zjednodušený editor kódu Visual Studio Code, jehož smyslem je poskytnutí nástrojů pro vývojáře, které jsou potřebné pro rychlé vytváření a ladění kódu. [11]

2.1.1 Překladač gcc

Tento překladač byl nejprve určen jako kompiler pro operační systém GNU. Jeho původním autorem je Richard Stallman, dnes je zastřešován skupinou zvanou Free Software Foundation, která si zakládá zejména na myšlence volně přístupného zdrojového kódu pro účely jeho prostudování a úpravy. V dnešní době gcc označuje sadu překladačů pro vícero programovacích jazyků (GNU Compiler Collection). [12]

2.2 Další vývojová prostředí

2.2.1 Eclipse

Toto vývojové prostředí je známé zejména pro vývoj v jazyce Java, každopádně jej lze využít i pro tvorbu v jazycích jako je C/C++, JavaScript, PHP a další. Zaštiťuje ho neziskové

sdružení zvané Eclipse foundation, které zahrnuje více než 275 členů. Jejich společnou myšlenkou je open source. [13]

2.2.2 Qt Creator

Jedná se o vývojové prostředí, které spadá pod multiplatformní framework s názvem Qt. Ten si zakládá hlavně na možnostech toho, že každý uživatel frameworku se může podílet na jeho dalším vývoji svými návrhy a myšlenkami. [14,15]

2.2.3 CLion

Vývojové prostředí od společnosti JetBrains, která například vyvíjí relativně nový programovací jazyk Kotlin. Jeho propojení s Projucerem je zatím v beta verzi. [16]

3 PŘEHLED POUŽITÝCH FUNKCÍ A TŘÍD

3.1 Z jazyka C/C++

3.1.1 Lambda funkce

Lambda funkce slouží zejména ke zjednodušení zápisu funkcí. Jsou vhodné tehdy, přiřazujeme-li jen několik málo řádků kódu k určité události, jako je například kliknutí na tlačítko, nebo také pokud víme, že danou funkci nebudeme v kódu volat na více místech. [17]

Je několik možných způsobů, jak deklarovat lambda funkci. V příkladech bude využit zejména nejkratší způsob zápisu – [captures]{body}, kde funkce nepotřebuje žádné vstupní argumenty ani nic nevrací.

Příklad takové funkce:

```
loadButton.onClick = [this] {
   auto name = processor.openFile();
   fileNameLabel.setText(name,dontSendNotification);
};
```

Tlačítku loadButton (třída TextButton z frameworku JUCE) přiřadíme lambda funkci, která reaguje na kliknutí na toto tlačítko (metoda onClick). Po kliknutí se zavolá funkce třídy AudioProcessor openFile, která slouží k otevření souboru a vrací jeho název. Ten poté uloží do proměnné fileName pomocí její členské funkce setText.

3.1.2 Enum

Výčtový datový typ enum je využíván zejména kvůli přehlednosti kódu a odstranění magických hodnot. Ty můžeme díky enumu reprezentovat jednoslovným názvem, který poté lze v kódu využívat na místech, kde bychom jindy použili jeho číselnou/textovou reprezentaci. [18]

Příklad využití:

```
enum {
   size = 512,
   count = 5
};
```

Nyní není potřeba v kódu psát dokola čísla 512 a 5, ale můžeme je nahrazovat řetězci size a count.

3.1.3 Struktury

Datová struktura je skupina dat, která jsou sdružena pod jedním názvem. Každá struktura má své členy, které mohou mít různé datové typy a délky. [19]

Příklad struktury:

```
struct Star {
   Point<float> coordinates;
   int currentLifeTime;
};
```

Poté můžeme přistupovat ke členům struktury Star pomocí tečky:

```
Star star;
star.currentLifeTime = 0;
```

Řádek výše provede nastavení člena struktury currentLifeTime typu int na 0.

3.1.4 auto

V moderním C++ může překladač automaticky odvodit datový typ výrazu. K tomu slouží 2 klíčová slova – auto a decltype. Pomocí auto můžeme vydedukovat například návratovou hodnotu funkce nebo výsledný datový typ výrazu. Při přiřazení auto x = 100; nebo auto y = getY(); o typu proměnné x rozhodne až překladač v době překladu. [20]

3.2 Z Frameworku JUCE

3.2.1 FileChooser

Tato třída slouží k vytvoření dialogového okna, ve kterém můžeme vybrat soubor k otevření nebo uložení. V praktické části bude použita při otevírání souboru pro přehrávání. K tomu budou využity členské funkce této třídy – browseFileToOpen, která otevře dialogové okno a umožní uživateli vybrat soubor, a getResult, která vrací zvolený soubor (pokud byl nějaký otevřen). [21]

3.2.2 AudioFormatManager, AudioFormatReader, AudioFormatReaderSource

Třídu AudioFormatManager můžeme využít pro vytvoření proměnné typu AudioFormatReader, která umí číst jednotlivé vzorky zvukových souborů. Třída se pokusí vybrat z dostupných formátů, pokud žádný vhodný nenajde, vrací nullptr. Toto vytvoření lze provést pomocí členské funkce createReaderFor(File audioFile), které předáme soubor, ze kterého se má vytvořit. [22,23]

Vytvořený reader lze dále předat třídě AudioFormatReaderSource, která ze souboru pomocí readera vytvoří přehrávatelný. [24]

3.2.3 AudioTransportSource

Z již vytvořeného přehrávatelného zdroje typu AudioFormatReaderSource můžeme dále vytvořit objekt typu AudioTransportSource, který za pomocí jeho členských funkcí start, stop a reset usnadní práci se zdrojem typu AudioFormatReaderSource. [25]

Pro interakci s uživatelem lze využít například třídy Slider, ComboBox nebo TextButton.

3.2.4 Slider

Posuvník lze využít ke změně hodnoty proměnných, můžeme volit mezi horizontálním, vertikálním a otáčivým – například pro úpravu frekvence, hlasitosti apod. [26]

Obrázek 1: Slider

3.2.5 ComboBox

Pomocí ComboBoxu lze zvolit ze seznamu několika možností. Tento seznam se skládá z textových řetězců, které jsou zobrazovány uživateli, kde každý z nich má svůj určitý identifikátor. [27]

Obrázek 2: ComboBox

3.2.6 Button

V JUCE můžeme najít různé typy tlačítek. ShapeButton, kterému můžeme nastavit určitý tvar, ImageButton, který obsahuje obrázek, ToggleButton, který lze zapínat a vypínat, ale nám postačí pouze jednoduché tlačítko s Textem – TextButton. [28]

Obrázek 3: TextButton

Ke grafické reprezentaci bodů a čar lze využít třídy Point a Path.

3.2.7 Point

Třída, která uchovává souřadnice x a y. Je vhodná zejména pro vykreslování ve funkci paint. [29]

3.2.8 Path

Ve třídě Path lze uchovávat křivky či úsečky, které lze poté například vykreslit funkcí paint. Začátek cesty se stanoví pomocí startNewSubpath, které předáme souřadnice bodu, ve kterém má cesta začít. Uzavření se provede pomocí closeSubPath. Mezi nimi lze volat několik různých funkcí, například lineTo, cubicTo nebo quadraticTo a tím udávat vzhled této “cesty“. [30]

3.2.9 Maths

V práci bude využita i třída maths, která nabízí různé datové typy (int32, int64…), funkce pro vyhledání minima/maxima (jmin, jmax) nebo třeba konstanty (π,e). [31]

4 PRÁCE SE ZVUKEM POMOCÍ KNIHOVY JUCE

V praktické části si ukážeme možnosti zpracování a úpravy zvuku pomocí frameworku. Ten například nabízí více možností k filtrování určitých frekvencí. V modulu DSP lze najít 3 různé typy filtrů – StateVariableFilter, IIRFilter a FIRFilter. Popis práce s nimi lze najít na již zmiňovaném YouTube kanále The Audi Programmer [7]. Krom toho lze také využít IIR filtr, který je zabudovaný již v základu frameworku a má o něco jednodušší přístup.

4.1 Filtr s nekonečnou impulzní odezvou

Je takový filtr, jehož impulsní charakteristika je nekonečná – odezva filtru na jednotkový impuls pro $n \rightarrow \infty$ *je* nenulová. [32]

U tříd IIRFilter a IIRCoefficients, lze pomocí statických funkcí vytvořit filtry s různými parametry. Například funkce makeLowPass vytvoří dolní propust.

V práci budou využity 3 typy filtru – horní, dolní a pásmová propust.

Horní propust je filtr, který propouští pouze vysoké frekvence a nízké frekvence utlumí. Dolní propust je pravým opakem. Propouští nízké frekvence a vysoké utlumí. Pásmová propust propouští frekvence v určitém kmitočtovém pásmu a frekvence pod ním a nad ním utlumí. [33]

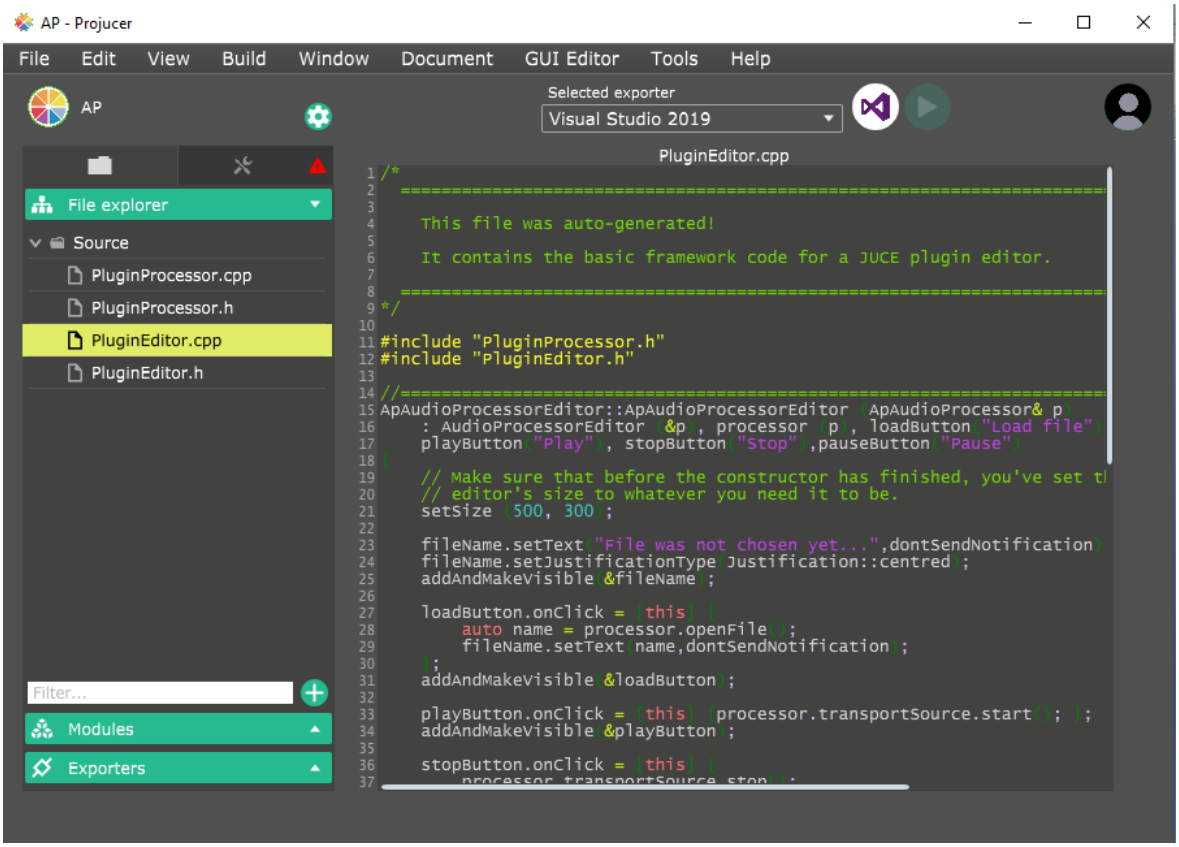
4.2 modul DSP

Framework JUCE umožňuje přidávání rozšiřujících modulů do projektu. V práci je zmiňován zejména modul DSP, který se zabývá zpracováním digitálních signálů. Umožňuje vytváření již zmiňovaných filtrů nebo také například implementuje Rychlou Fourierovu transformaci.

4.3 Rychlá Fourierova transformace

Rychlá Fourierova transformace je velmi efektivní algoritmus pro výpočet diskrétní Fourierovi transformace hlavně díky tomu, že eliminuje některé zbytečné aritmetické operace potřebné k provedení výpočtu. K výpočtu DSP je zapotřebí provést $O(N^2)$ operací, zatím co u FFT se počet sníží na přibližně $O(N * logN)$. Takže například pro DFT s 8 body by bylo zapotřebí provést 64 nutných operací, zatímco u FFT by stačilo okolo 22 operací. [33]

5 PRÁCE S PROJUCEREM



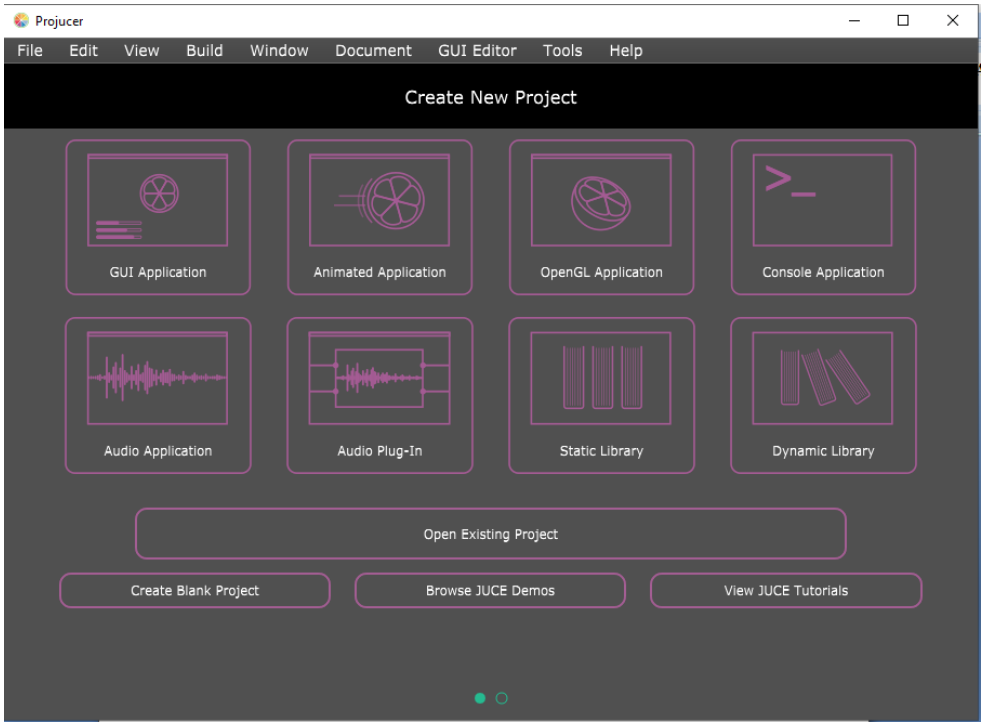
Obrázek 4: Projucer

Práce s Projucerem se nijak neliší od jiných aplikací či IDE. Při spuštění se automaticky otevře poslední spuštěný projekt.

Ve File Exploreru lze upravovat jednotlivé soubory projektu. V horní části lze zvolit jiný způsob otevření projektu. Zde je vidět možnost zobrazení ve Visual Studio, ve kterém můžeme program spustit pomocí tlačítka s logem tohoto IDE.

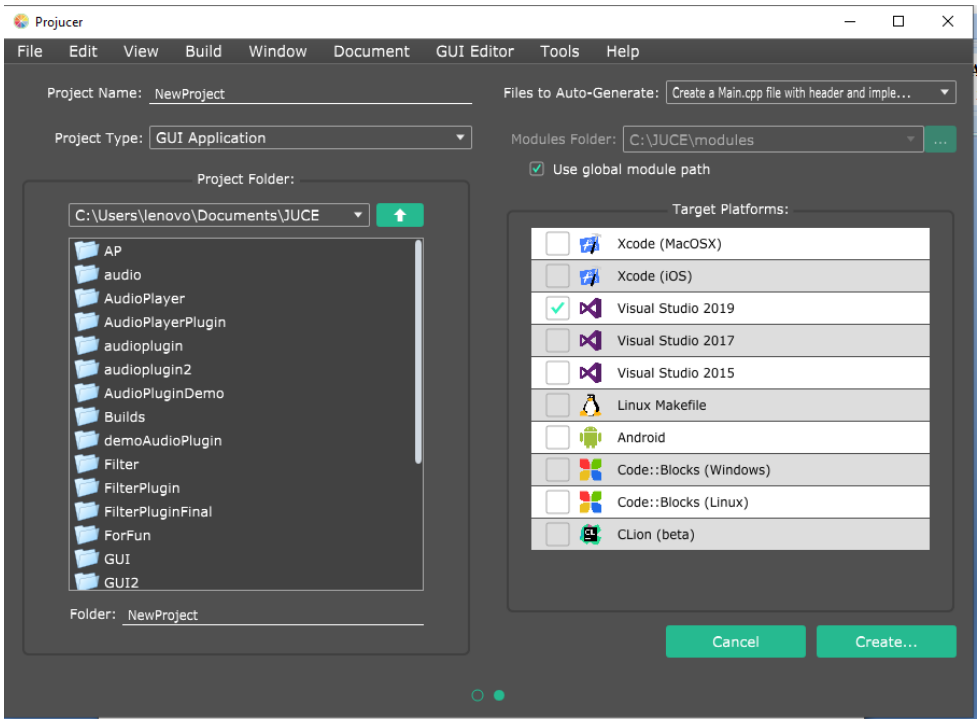
5.1 Založení nového projektu

V horním menu Projuceru vybereme *File* > *New Project*. Zobrazí se následující okno:



Obrázek 5: Založení nového projektu 1

Kromě 8 různých typů aplikací lze také otevřít již existující projekty, založit prázdný projekt a otevřít DEMO projekty nebo se odkázat na stránku s tutoriály.



Obrázek 6: Založení nového projektu 2

Po vybrání typu aplikace, kterou chceme tvořit, se zobrazí okno, ve kterém můžeme zvolit, v jakém prostředí chceme soubory upravovat. Dále můžeme vybrat, které soubory chceme, aby za nás Producer vygeneroval – tím získáme určitý základ naší aplikace, který můžeme dále rozvíjet. V práci jsou využívány šablony typu Audio Plug-In a Animated Application.

5.2 Základní struktura pluginu v JUCE

Při založení nového projektu typu Audio Plug-In máme možnost si nechat vygenerovat jeho základní kostru. Ta je rozdělena do dvou tříd – PluginEditor a PluginProcessor.

Třída PluginEditor slouží k úpravě vizuální stránky našeho pluginu. V jejím konstruktoru je potřeba zavolat funkci setSize(int x, int y), kde x je šířka okna aplikace a y jeho výška, jinak se okno aplikace nezobrazí. Dále zde přidáváme jednotlivé prvky do naší aplikace pomocí funkce addAndMakeVisible(&nazevPrvku), můžeme nastavit i různé vlastnosti těchto prvků nebo jim přiřadit listenery. Vyjma destruktoru najdeme dále v této třídě dvě funkce. Ta s názvem paint(Graphics& g) slouží k vykreslení pozadí, či jiných grafických prvků do okna aplikace pomocí funkcí které obsahuje ukazatel g – například g.setFont(15.0f). Funkce s názvem resized slouží k rozložení prvků aplikace, jako jsou různá tlačítka nebo posuvníky. Těm můžeme nastavit vlastnost setBounds(int odsazeníZleva, int odsazeníZhora, int šířkaPrvku, int výškaPrvku) a tím dát aplikaci vědět, jak mají být jednotlivé prvky rozloženy.

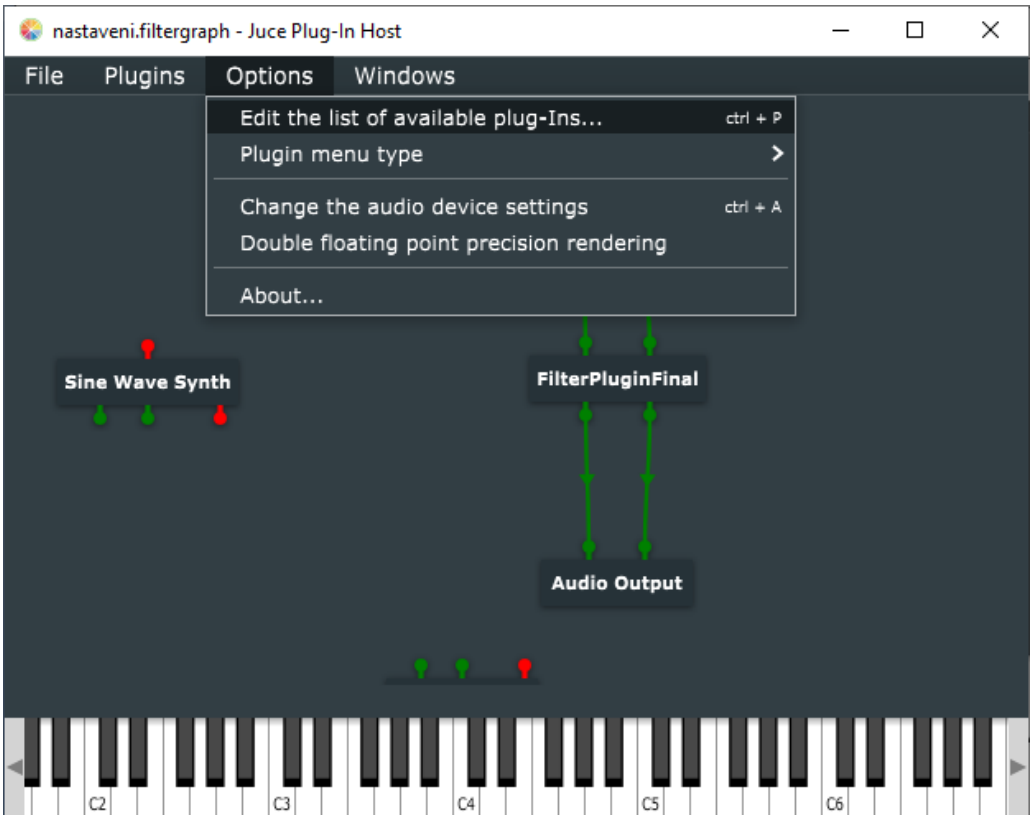
Ve třídě PluginProcessor můžeme, kromě tvoření vlastních, také využívat již předdefinované funkce pro obsluhu našeho pluginu. Některé funkce, jako třeba prepareToPlay, mají garantované, kdy budou volány. Konkrétně tato funkce je volána vždy před tím, než má začít přehrávání audia. Funkce processBlock(AudioBuffer<float>& buffer, MidiBuffer& midiMessages) slouží k připravení další části přehrávaného audia. Proměnná buffer obsahuje počet kanálů, který odpovídá maximu počtu vstupních nebo výstupních kanálů. Pokud procesor rozpozná nějaké změny na MIDI vstupních zařízení (a jsou-li nějaká připojena), je možné na tyto události reagovat, jelikož se ukládají do pole s názvem midiMessages. Jejich zpracování závisí poté na programátorovi. Po pozastavení přehrávání se volá funkce releaseResources, která uvolní veškeré již nepotřebné zdroje. [34]

5.3 Práce s pluginy

Spolu s Projucerem se stáhne i několik ukázkových projektů. Z nich je velmi pomocný ten s názvem Audio Plugin Host. Ten lze najít ve složce *\JUCE\extras*. Po jeho spuštění je možné propojovat jak některé demo projekty, tak Vámi vytvořené pluginy.

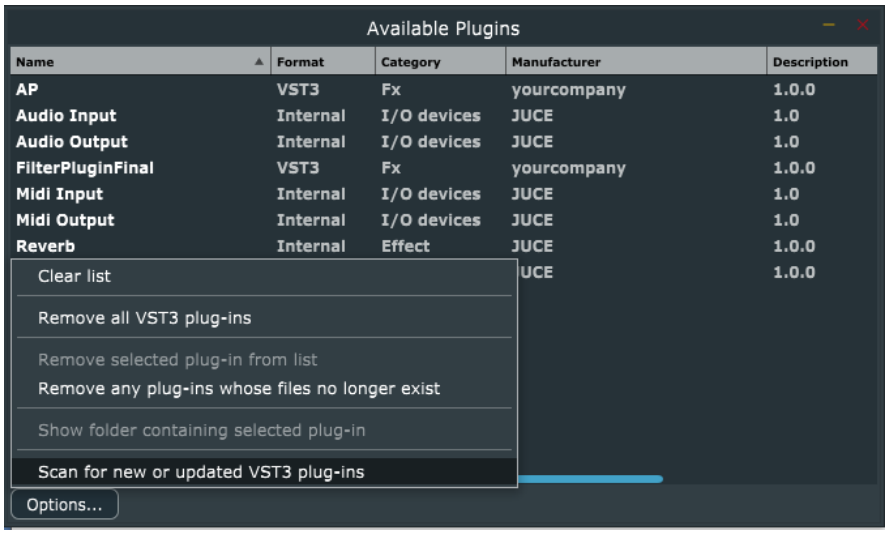
5.3.1 Přidání nových pluginů

Demo projekty jsou dostupné hned po spuštění. Pokud chce uživatel přidat jím vytvořené programy, stačí pouze přidat cestu k onomu souboru. Po kliknutí na *Options > Edit the list of available plug-Ins…* se zobrazí seznam dostupných pluginů.



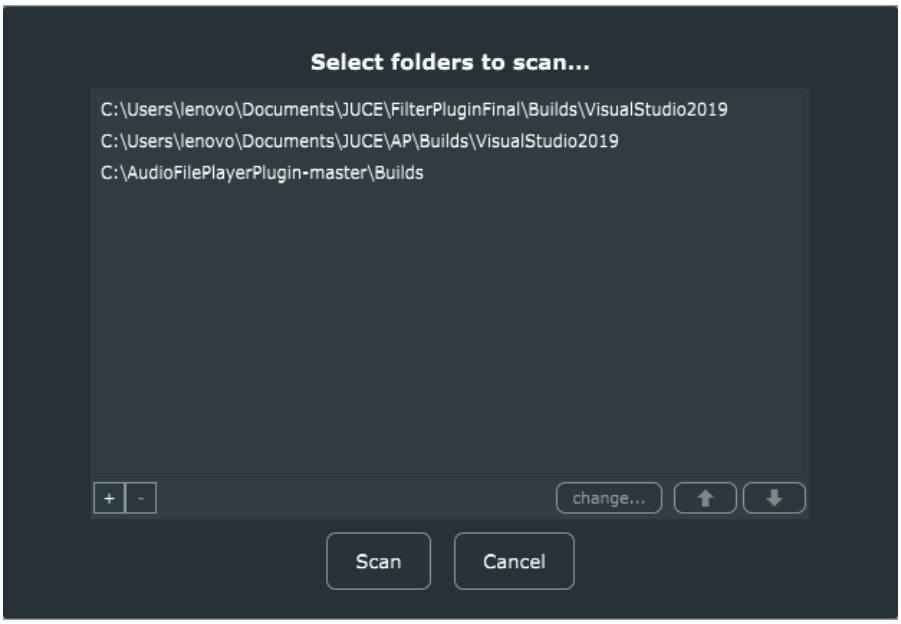
Obrázek 7: doplnění cest k dostupným pluginům

Kliknutím na *Options…* > *Scan for new updated VST3 plug-ins* se zobrazí již přidané cesty k pluginům.



Obrázek 8: Zjištění dostupnosti pluginů podle zadaných cest

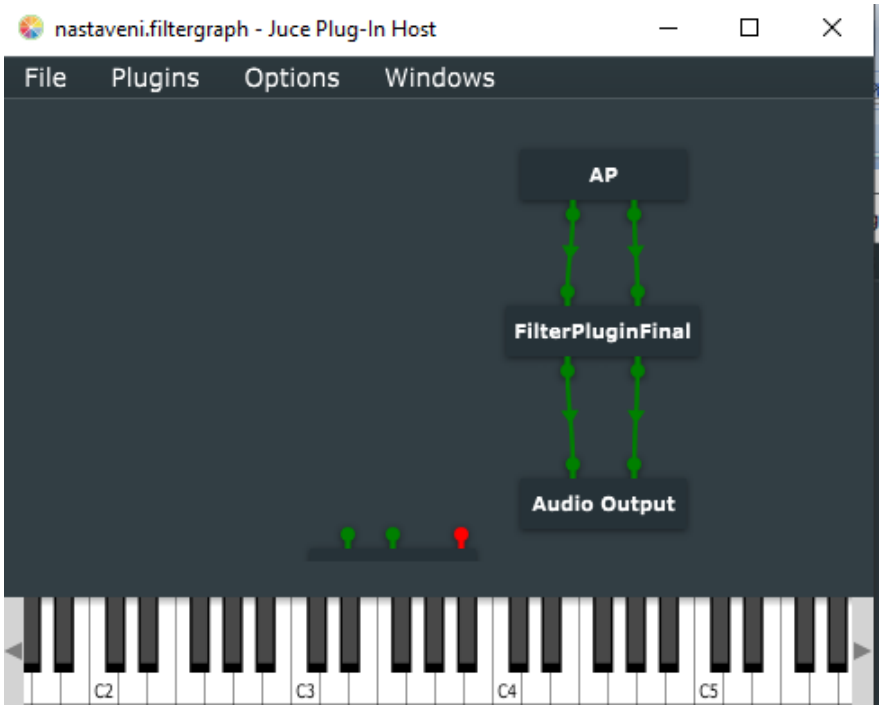
Kliknutím do levého dolního roku na + lze přidat cesty k dalším uživatelem vytvořeným pluginům. Po kliknutí na Scan se aplikace pokusí vyhledat spustitelné soubory. Je-li nalezen nějaký plugin, objeví se v seznamu všech dostupných a lze jej přidat do aplikace po kliknutí pravým tlačítkem a vybrání ze složky, která má název podle názvu položky Plugin manufacturer v nastavení projektu.



Obrázek 9: Okno s cestami pro vyhledání pluginů

5.3.2 Propojování jednotlivých pluginů v Audio Plugin Hostu

Vstupy a výstupy jednotlivých pluginů (mají-li nějaké) můžeme jednoduše propojovat mezi sebou. Na obrázku níže je zobrazeno takové propojení, kdy jsou všechny pluginy typu stereo (dva kanály). V propojení jsou zobrazeny šipky, aby bylo poznat odkud kam jsou jednotlivé pluginy propojeny. Výstup přehrávače AP se stává vstupem filtru FilterPluginFinal, který je poté propojen s audio výstupem.



Obrázek 10: Propojení v Audio Plugin Hostu

5.4 Animovaná aplikace

Dalším typem aplikace, kterou lze v JUCE vytvořit je aplikace animovaná. Po založení projektu tohoto typu získáme jinak předdefinovanou strukturu než u aplikace typu plugin. Jedná se o určitý základ, se kterým lze například velmi jednoduše tvořit GUI aplikací.

Soubor Main.cpp se stará o spouštění a správné ukončení aplikačního okna. Pro samotnou kostru aplikace je předdefinovaná také dvojice souborů MainComponent.h a MaincComponent.cpp, ve kterých jsou již předdefinované metody update, paint a resized.

V konstruktoru souboru .cpp zavoláme funkci setFramesPersSecond(int framesPerSecond). Počet snímků za vteřinu nám stanoví, kolikrát za vteřinu se nám budou volat funkce update a paint. Funkci update lze použít pro aktualizaci veškerých hodnot, paint slouží

k vykreslování komponentů aplikace. Poslední předdefinovaná funkce resized je, jak jde již podle názvu poznat, aplikací volána po změně velikosti okna.

II. PRAKTICKÁ ČÁST

6 VYPRACOVANÉ ÚKOLY

6.1 Úvod

Jak již bylo zmíněno, cílem práce je zhodnocení využitelnosti tohoto frameworku a navržení úkolů, které studentům přiblíží a vizualizují některé probírané problematiky. První 3 úkoly na sebe navazují a lze je propojit v Audio Plugin Hostu. Jedná se o přehrávač, filtr a vizualizér.

Audio přehrávač umí otevřít soubor, spustit jej, pozastavit nebo úplně zastavit jeho přehrávání. Dále může měnit hlasitost přehrávání.

Filtr po připojení k určitému vstupu (přehrávač, mikrofon…) je schopen filtrovat frekvence ve třech režimech – dolní, horní nebo pásmová propust. Lze u něj nastavit, jaké frekvence propouští a jaké ne a velikost ozvěny.

U vizualizéru můžeme po připojení nějakého vstupu vybrat mezi zobrazením v reálném čase nebo jako spektrogram.

Poslední úkol již není tvořen jako plugin, ale jedná se o animovanou aplikaci. Reprezentuje zjednodušený model Sluneční soustavy.

Úkoly jsou vytvářeny ve verzi JUCE 5.4.7.

6.2 Úkol č. 1 – Audio přehrávač

Prvním úkolem, který by měl být velice jednoduchý, je vytvoření audio přehrávače. Vychází z tutoriálů, které jsou k dispozici přímo na stránkách frameworku. V něm [34] je tvořena Audio Aplikace, ta ale není pro naše užití vhodná, jelikož bychom ji nebyli schopni jednoduše přidat do Audio Plugin Hosta a dále ji propojovat s dalšími pluginy. Z toho důvodu budeme tvořit aplikaci typu Plug-In, kde se nám automaticky vygenerují výše zmiňované třídy PluginEditor a PluginProcessor.

6.2.1 Stanovení zadání úkolu

Pomocí frameworku JUCE vytvořte audio přehrávač (skrz šablonu typu Plug-In), který bude mít možnost přehrávat mp3 soubory. Měl by být schopen otevřít soubor a ten poté přehrát pomocí tlačítka Play. Dále bude možné pozastavit přehrávaný soubor tlačítkem Pause nebo úplně zastavit jeho přehrávání tlačítkem Stop. Pokud uživatel pozastaví přehrávaný soubor

tlačítkem Pause, může poté pokračovat v přehrávání stisknutím tlačítka Play. Dále bude možné měnit hlasitost přehrávaného souboru – úroveň hlasitosti 0-100.

K realizaci přehrávače využijte třídy AudioFormatManager, AudioFormatReaderSource, AudioTransportSource, Slider, Label, TextButton.

6.2.2 Řešení

Založíme si nový projekt typu Audio Plug-in (viz kapitola 5.1). V souboru PluginEditor.h přidáme do části pod klíčovým slovem private 4 tlačítka typu TextButton, které pojmenujeme loadButton, playButton, stopButton a pauseButton. Dále přidáme posuvník (Slider), který pojmenujeme volumeSlider, a popisek (Label), který bude zobrazovat název přehrávaného souboru a pojmenujeme jej fileNameLabel.

6.2.3 Vzhled aplikace

Poté se můžeme pustit na vizuální stránku přehrávače. Jelikož není podstatou úkolu stylizace přehrávače, nebudeme jí věnovat příliš velkou pozornost. V souboru PluginEditor.cpp využijeme funkci resized, ve které si nastavíme rozložení jednotlivých prvků. Využijeme k tomu funkci setBounds(int souřadniceX, int souřadniceY, int šířka, int výška), kterou mají implementovanou všechny naše komponenty (tlačítka, slider i label). Dále využijeme funkci getWidth, která vrací aktuální šířku aplikačního okna v pixelech.

Pomocí funkce setBounds tedy nastavíme prvkům loadButton, fileNameLabel, playButton, pauseButton, stopButton a volumeSlider pozici v okně aplikace. Pokud to uděláme ve funkci resized, bude se nám se změnou velikosti okna měnit i velikost prvků přehrávače.

```
loadButton.setBounds(10, 10, getWidth() - 20, 30);
fileNameLabel.setBounds(10, 50, getWidth() - 20, 30);
playButton.setBounds(10,90, getWidth() - 20 ,30);
pauseButton.setBounds(10, 130, getWidth() - 20, 30);
stopButton.setBounds(10, 170, getWidth() - 20, 30);
volumeSlider.setBounds(10, 210, getWidth() - 20, 30);
```

Pokud bychom nyní zkusili spustit aplikaci, tak přesto, že jsme již nastavili velikost všem komponentám přehrávače, nezobrazí se nám žádný.

Aby byly jednotlivé komponenty viditelné, je potřeba v konstruktoru zavolat funkci addAndMakeVisible(Component*child), do které předáme ukazatel na komponent, který chceme zviditelnit.

```
ApAudioProcessorEditor::ApAudioProcessorEditor (ApAudioProcessor& p)
   : AudioProcessorEditor (&p), processor (p), loadButton("Load file"),
   playButton("Play"), stopButton("Stop"),pauseButton("Pause")
{
   setSize (500, 300);
   setResizable(true, true);

}
```

Po pozdějším přidání komponent pomocí funkce addAndMakeVisible a stanovení velikosti okna funkcí setSize budou jednotlivé prvky aplikace viditelné. Funkce setSize synchronně volá funkci resized a musí být v konstruktoru každé aplikace, jinak spadne hned po spuštění. Při vytváření tlačítek můžeme předat řetězec, který má být na tlačítku zobrazen. Proto každému z nich přidáme jeho popisek, jak jde vidět na obrázku výše, a to například pomocí loadButton(“Load File“). Jiným ekvivalentem by bylo nastavení textu pomocí funkce setButtonText(String text) – např. loadButton.setButtonText(“Load file“).

6.2.4 Otevření souboru

Nyní se vrhneme na samotnou kostru přehrávače, která je popsaná v tutoriálu [35], ze kterého zadání vychází. Základní pointou je implementace otevření souboru a uložení jej do objektu třídy AudioTransportSource [25], který lze poté jednoduše spustit, pozastavit a podobně.

Do souboru PluginProcessor.h přidáme, za již předem vygenerované funkce, komponenty a funkce, které budeme potřebovat

```
String openFile();
void updateGainValue(double value);
```

Funkce openFile bude sloužit jak jinak než k otevření souboru pro přehrávání. Bude vracet řetězec s názvem souboru, který poté uložíme do labelu fileNameLabel. Funkce updateGainValue využijeme pro úpravu hlasitosti.

Dále deklarujeme objekty tříd, díky kterým budeme moci otevírat soubory:

```
AudioFormatManager manager;
std::unique_ptr<AudioFormatReaderSource> readerSource;
AudioTransportSource transportSource;
```

Pro správnou funkčnost třídy AudioFormatManager je potřeba určit, s jakými formáty může pracovat. Nám stačí základní formáty, a proto zavoláme v implementačním souboru AudioProcessor.cpp – manager.registerBasicFormats a tím managera nastavíme.

V AudioProcessor.cpp nejdříve vytvoříme tělo funkce openFile:

```
String ApAudioProcessor::openFile() {

   FileChooser chooser("Select a file to play...",
      {},
      "*.mp3");
   String fileName = "No file was chosen...";


   if (chooser.browseForFileToOpen())
   {
      auto file = chooser.getResult();
      auto* reader = manager.createReaderFor(file);


      if (reader != nullptr)
      {
         fileName = file.getFileName();
         std::unique_ptr<AudioFormatReaderSource> newSource(new
AudioFormatReaderSource(reader, true));
         transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);
         readerSource.reset(newSource.release());
      }
   }
   return fileName;
}
```

K otevření souboru si vytvoříme instanci třídy FileChooser, té předáme řetězec, který se má zobrazit v titulku okna. Druhou položkou můžeme nastavit, která složka se má zobrazit jako výchozí. Jelikož nechceme zadávat žádnou výchozí složku vyplníme pouze prázdné složené závorky. Poslední položkou jsou soubory, které chceme, aby bylo možno vybrat.

Vyplníme pouze soubory typu mp3. V případě více různých souborů je můžeme oddělit středníkem – **.mp3;*.wav*. Proměnnou fileName nastavíme na řetězec, který hlásí, že uživatel nevybral žádný soubor – v případě, že nějaký vybere, tak později řetězec nahradíme názvem souboru, pokud se výběr nezdaří, tak jej vyplníme chybovou hláškou.

Při zavolání funkce otevřeme dialogové okno pomocí chooser.browseFileToOpen. Tato funkce je typu boolean a vrací true pokud uživatel vybere nějaký soubor. Pokud okno zavře, aniž by vybral soubor, tak se vrací false. Při úspěšném vybrání souboru můžeme pokračovat dál a funkcí chooseru getResult si uložíme vybraný soubor do proměnné file a vytvoříme si na něj ukazatel reader, který lze pomocí funkce manažera přečíst createReaderFor(File file).

Pokud se reader úspěšně vytvoří, tak do proměnné fileName uložíme název souboru a vytvoříme si unikátní ukazatel na objekt typu AudioFormatReaderSource, kterému předáme readera a druhý argument funkce nastavíme na true, čímž se zajistí smazání readera pokud smažeme i ukazatel typu AudioFormatReaderSource.

Vytvořenou proměnnou newSource předáme třídě TransportSource pomocí funkce setSource. Té předáme ukazatel na zdroj, který chceme přehrávat, pomocí funkce get. Dále můžeme nastavit, má-li se na pozadí dopředu načítat další část souboru. Pokud ano, nastavíme jako druhý argument velikost bufferu a v třetím argumentu předáme ukazatel na proměnnou typu TimeSlicedThread. Poslední argument předáme velikost vzorkovací frekvence, kterou získáme z readera zapomocí ukazatele na jeho položku sampleRate. Po provedení tohoto řádku ještě předáme lokální proměnnou newSource do proměnné readerSource pomocí předání vlastnictví funkcí std::unique_ptr::release().

Jako poslední pouze vrátíme název otevřeného souboru pomocí return fileName.

6.2.5 Úprava hlasitosti

V dokumentaci ke třídě AudioTransportSource [25] se můžeme dozvědět, že funkce setGain vyžaduje číslo typu float. Toto číslo určuje, čím se mají vynásobit výstupní vzorky zvukové stopy. Dokumentace uvádí, že pro hodnotu 0.5 je to -6 dB, pro 1.0 0 dB apod. Z toho, že se mají vzorky násobit určitým číslem vyplívá, že při vynásobení 0 bude hlasitost přehrávače plně ztlumena. Hodnotu 1.0 budeme brát jako 100% hlasitosti (jelikož výstup nebude nijak upravován).

```
void ApAudioProcessor::updateGainValue(double value) {
   transportSource.setGain(value / 100);
}
```

Vezmeme v potaz, že chceme hlasitost na slideru udávat mezi 0 – 100 % a funkce vyžaduje desetinné číslo mezi 0-1, tudíž jednoduše vezmeme hodnotu slideru a tu podělíme 100 a nastavíme zisk výstupu na tuto hodnotu.

6.2.6 Využití funkcí pluginu k přehrávání

Nyní se ukáže síla frameworku, konkrétně šablony pluginu. Jelikož předpokládá, že budeme nějak pracovat s audiem, má připravené funkce a propojené jejich volání v moment, kdy je to potřeba.

Ve funkci prepareToPlay pouze zavoláme transportSource.prepareToPlay(), které předáme hodnoty samplesPerBlock a sampleRate a tím nastavíme transportSource do stavu, který lze přehrát.

```
void ApAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
   transportSource.prepareToPlay(samplesPerBlock, sampleRate);
}
```

Ve funkci releaseResources pouze zavoláme transportSource.releaseResources() a tím, jak již název napovídá uvolníme zdroje této třídy. Je zaručeno že funkce prepareToPlay se zavolá při spuštění programu minimálně jednou pokud se AudioSource (u nás podtřída TransportSource) nachází v nepřipraveném stavu. [36]

Platí to i pro funkci releaseResources, ta se zase volá při ukončení aplikace. To si ostatně můžete ověřit v debuggeru.

```
void ApAudioProcessor::releaseResources()
{
   transportSource.releaseResources();
}
```

Dále je potřeba doplnit funkci, která zpracovává jednotlivé bloky audio souboru. Při založení projektu se nám již určitým způsobem připravila funkce processBlock. Nám ale stačí pouze vyčistit buffer od případného nepořádku a poté buffer předat do transportSource pomocí funkce getNextAudioBlock, která ale očekává typ AudioSourceChannelInfo, tudíž buffer ještě přetypujeme a funkci upravíme.

```
void ApAudioProcessor::processBlock (AudioBuffer<float>& buffer, MidiBuffer&
midiMessages)
{
   auto totalNumInputChannels  = getTotalNumInputChannels();
   auto totalNumOutputChannels = getTotalNumOutputChannels();

   for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
      buffer.clear (i, 0, buffer.getNumSamples());

   transportSource.getNextAudioBlock(AudioSourceChannelInfo(buffer));

}
```

Na straně procesoru je potřeba nastavit přehrávání od začátku souboru pomocí funkce setPosition. Tudíž přidáme do konstruktoru transportSource.setPosition(0.0).

6.2.7 Propojení Editoru a Processoru

Máme již nachystanou funkci přehrávače, ale tu je potřeba propojit s naším UI. Vrátíme se tedy do souboru PluginEditor.cpp a nastavíme funkce jednotlivým tlačítkům.

Popisku fileNameLabel nastavíme text, který bude indikovat, že zatím nebyl vybrán žádný soubor pomocí funkce setText, kde prvním argumentem je text, který se má zobrazit a druhým typ notifikace. My na toto nastavení nepotřebujeme nijak reagovat, tudíž zde nastavíme argument na dontSendNotification. Druhý řádek nastaví zarovnání textu v popisku na střed.

```
fileNameLabel.setText("File was not chosen yet...",dontSendNotification);
fileNameLabel.setJustificationType(Justification::centred);
addAndMakeVisible(&fileNameLabel);
```

Poté přiřadíme eventu tlačítka loadButton onClick anonymní funkci, která zavolá funkci PluginProcesoru openFile a vrátí nám název souboru, který uživatel zvolil. Tento název uložíme do labelu opět funkcí setText.

```
loadButton.onClick = [this] {
    auto name = processor.openFile();
    fileNameLabel.setText(name,dontSendNotification);
};
addAndMakeVisible(&loadButton);
```

Pro spuštění přehrávání je potřeba zavolat funkci start třídy AudioTransportSource, do které jsme si zjednodušeně řečeno připravili soubor, který chceme přehrát. Tlačítku playButton a jeho eventu onClick tedy přiřadíme anonymní funkci pomocí reference processoru, kterou nám připravila šablona (processor.transportSource.start()).

```
playButton.onClick = [this] {processor.transportSource.start(); };
addAndMakeVisible(&playButton);
```

To samé provedeme pro tlačítka stop a pause s tím rozdílem, že voláme funkci třídy AudioTransportSource s názvem stop. U tlačítka stop ještě nastavíme pozici přehrávaného audia na 0.0, aby při případném opětovném přehrávání se začalo znovu od začátku.

```
stopButton.onClick = [this] {
    processor.transportSource.stop();
    processor.transportSource.setPosition(0.0);
};
addAndMakeVisible(&stopButton);


pauseButton.onClick = [this] {processor.transportSource.stop(); };
addAndMakeVisible(&pauseButton);
```

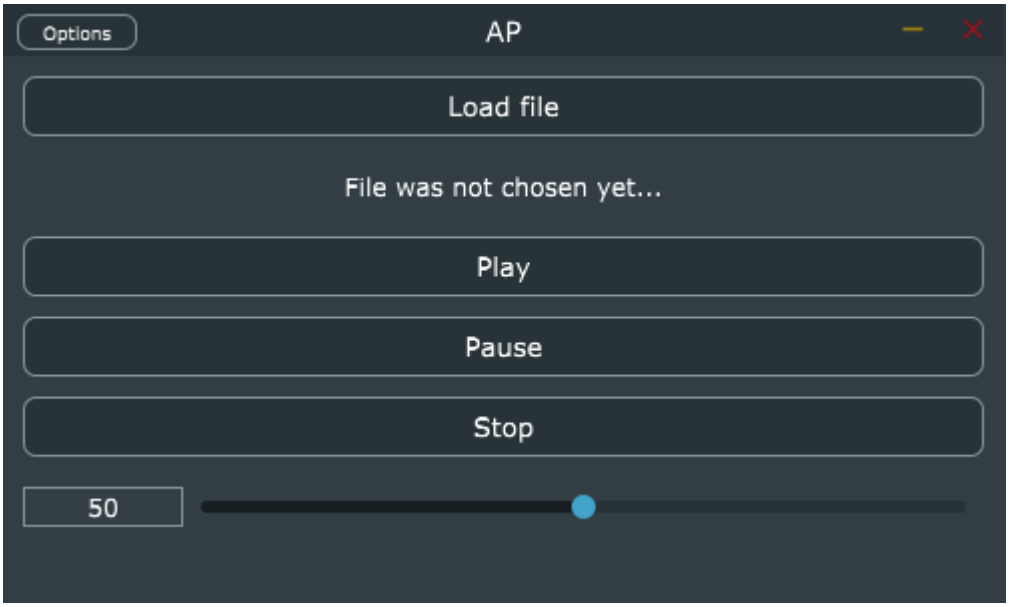
Jako poslední je potřeba moci nastavit hlasitost přehrávání. Pomocí setRange můžeme nastavit rozmezí hodnot posuvníku, kde první parametr je minimální hodnota, druhý maximální hodnota a třetí krok, po kterém se má hodnota měnit.

```
volumeSlider.setRange(0,100,1);
```

Funkci pro změnu hlasitosti již máme připravenou, tudíž ji pouze přiřadíme eventu onValueChange, který je volán vždy při změně hodnoty posuvníku. Pomocí funkce setValue mu můžeme nastavit hodnotu v rozmezí, které jsme zadali pomocí funkce setRange.

```
volumeSlider.onValueChange = [this]
{processor.updateGainValue(volumeSlider.getValue()); };
volumeSlider.setValue(50);
addAndMakeVisible(&volumeSlider);
```

Tím je audiopřehrávač hotov a měl by vypadat tak, jako na obrázku níže:



Obrázek 11: Audiopřehrávač

6.3 Úkol č. 2 – Tvorba filtru

Když máme již hotový audio přehrávač, vytvoříme k němu jednoduchý filtr. Pro ukázku nám bude stačit implementovat horní, dolní a pásmovou propust. Ve frameworku můžeme najít několik různých filtrů. Využijeme ten nejjednodušší z nich – filtr s nekonečnou impulzní odezvou IIRFilter. Jeho funkčnost lze zajistit velmi jednoduše pomocí šablony pluginu. Další filtry můžeme najít například v modulu frameworku s názvem DSP, každopádně práce s nimi je o něco složitější.

6.3.1 Stanovení zadání úkolu

Vytvořte audio filtr, který bude schopný po propojení s audio přehrávačem pomocí Audio Plugin Hosta filtrovat přehrávaný soubor přehrávače. Umožněte uživateli vybrat mezi horní, dolní a pásmovou propustí a nastavovat úroveň propouštěné frekvence a ozvěny. Pro možnou frekvenci zvolte rozsah 0 Hz – 20 KHz a pro ozvěnu hodnoty 0.1 – 5. K realizaci využijte třídu IIRFilter. Pro nastavování hodnot lze využít třídy Slider a ComboBox.

6.3.2 Řešení

Opět vygenerujeme šablonu typu Plug-In, aby bylo možné aplikaci spouštět v Audio Plugin Hostovi.

6.3.3 Vzhled aplikace

O vizuální stránku aplikace se stará třída PluginEditor. Pro nastavení rozsahu je ideální použít Slider, proto si vytvoříme 2 instance této třídy, jednu pro frekvenci a druhou pro ozvěnu. Pro výběr typu filtrace lze využít třídy ComboBox.

Nejprve si nastavíme velikost okna na 400x150 pixelů pomocí setSize a zakážeme uživateli možnost změny jeho velikosti pomocí funkce setResizable(false, false). Poté přidáme do naší aplikace Slidery a ComboBox, pomocí funkce addAndMakeVisible v konstruktoru, Ve funkci resized nastavíme rozložení jednotlivých prvků funkcí setBounds.

```
frequency.setBounds(10,10,getWidth()-20,30);
resonance.setBounds(10,50,getWidth()-20,30);
filterType.setBounds(10,90, getWidth()-20, 30);
```

Poté si vytvoříme funkci pro aktualizaci filtru, kterou pojmenujeme updateFilter. Touto funkcí budeme upravovat typ a hodnoty filtru. Pro posuvník s frekvencí si nastavíme rozsah 0,1 – 20000, po kroku 100, a hodnotu nastavíme na 3000. Eventu Slideru onValueChange přiřadíme anonymní funkci, která zavolá funkci sloužící pro aktualizaci filtru.

```
frequency.setRange(0.1, 20000.0, 100.0);
frequency.setValue(3000.0);
frequency.onValueChange = [this] {
    updateFilter();
};
```

Je důležité, aby spodní mez frekvence nebyla rovna 0, zabudovaný filtr by totiž nepracoval správně a způsoboval by padání programu.

```
resonance.setRange(0.1,5.0,0.1);
resonance.setValue(1.0);
resonance.onValueChange = [this]{
    updateFilter();
};
```

U ozvěny si nastavíme rozsah od 0.1 do 5.0 po kroku 0.1 a nastavíme počáteční hodnotu na 1.0. Opět přiřadíme funkci pro aktualizaci filtru.

```
filterType.addItem("low pass", 1);
filterType.addItem("high pass", 2);
```

```
filterType.addItem("band pass", 3);
filterType.setSelectedId(1);
filterType.onChange = [this] {
    updateFilter();
};
```

Do ComboBoxu nejprve přidáme pomocí funkce addItem typy filtru, kde prvním argumentem je text možnosti, který bude ComboBox zobrazovat a druhým jeho identifikátor. ComboBox nastavíme na filtr typu dolní propust a opět přiřadíme funkci pro aktualizaci filtru a tím máme stranu editoru téměř hotovou.

6.3.4 Filtr

Ve třídě PluginProcessor si přidáme IIRFilter a 2 pomocné proměnné pro jeho nastavení. První typu double bude udávat vzorkovací frekvenci vstupního audia (z přehrávače) a druhá, typu integer, počet vzorků, které má filtr zpracovat.

Využijeme funkci šablony prepareToPlay a uložíme si vzorkovací frekvenci a počet vzorků, který budeme filtru předávat.

```
void FilterAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    actualSampleRate = sampleRate;
    samples = samplesPerBlock;
}
```

Poté můžeme přejít na další funkci, která se stará o zpracování jednotlivých vzorků. Díky šabloně máme nachystanou funkci processBlock. Ta nejprve pročistí buffer od případných anomálií a poté umožní v cyklu pracovat s daty z bufferu. My zde pouze přidáme 2 řádky. Před druhým cyklem resetujeme filtr a tím jej připravíme na zpracování dalších dat. Pak v cyklu zavoláme funkci filtru processSamples, které přidáme ukazatel na data bufferu a počet vzorků, které má zpracovat.

```
void FilterAudioProcessor::processBlock (AudioBuffer<float>& buffer, MidiBuffer&
midiMessages)
{
    auto totalNumInputChannels  = getTotalNumInputChannels();
    auto totalNumOutputChannels = getTotalNumOutputChannels();

    for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
        buffer.clear (i, 0, buffer.getNumSamples());

    filter.reset();

    for (int channel = 0; channel < totalNumInputChannels; ++channel)
    {
        auto* channelData = buffer.getWritePointer (channel);

        filter.processSamples(channelData, samples);
    }
}
```

Nyní máme nachystanou funkcionalitu filtru, ale zatím jsme mu nedali nijak vědět, jakým způsobem má data filtrovat. Proto si vytvoříme funkci setFilterCoefficients, která bude nastavovat filtr na požadovaný typ. Tu pak budeme volat z editoru funkcí updateFilter.

```
void FilterAudioProcessor::setFilterCoefficients(double frequency,double resonance,int option) {

    IIRCoefficients coefficients;

    switch (option) {
    case 1:
        coefficients = IIRCoefficients::makeLowPass(actualSampleRate, frequency, resonance);
        break;
    case 2:
        coefficients = IIRCoefficients::makeHighPass(actualSampleRate, frequency, resonance);
        break;
    case 3:
        coefficients = IIRCoefficients::makeBandPass(actualSampleRate, frequency, resonance);
        break;
    }

    filter.setCoefficients(coefficients);
}
```

Jak lze vidět, podle předaných hodnot pouze rozhodneme, na jaký typ se má filtr nastavit a podle toho si vytvoříme jeho koeficienty. Ty poté nastavíme filtru pomocí funkce setCoefficients.

6.3.5 Propojení Editoru a Processoru

Jediné, co zbývá, je doplnit tělo funkce v editoru. Pomocí příslušných funkcí si zjistíme hodnotu posuvníků a ComboBoxu a ty poté předáme funkci processoru setFilterCoefficients.

```
void FilterAudioProcessorEditor::updateFilter() {
    processor.setFilterCoefficients(
        frequency.getValue(),
        resonance.getValue(),
        filterType.getSelectedId()
    );
}
```

Pak už jen zbývá aplikaci otestovat společně s přehrávačem pomocí Audio Plugin Hosta.

Obrázek 12: Filtr

6.4 Úkol č. 3 – Vizualizace audio stop

S ohledem na to, že máme již vytvořen audio přehrávač i filtr a máme možnost slyšet úpravu audio stop, bylo by pěkné mít možnost tuto změnu i vidět. V tutoriálech na stránkách frameworku můžeme najít několik příkladů. Samotný framework nabízí dokonce hotovou komponentu AudioVisualiserComponent [37], ale její použití by bylo příliš jednoduché a nic neříkající, protože zobrazuje pouze amplitudu vzorků.

S přihlédnutím k tomu, že práce je cílena pro výuku programování, bylo by dobré zahrnout i něco, s čím se studenti mohli setkat ve výuce. V tutoriálech lze najít 2 velmi podobné způsoby řešení pomocí FFT. První [38] z nich zobrazuje výsledek jako spektrogram, kde lze vidět v čase (osa x) se měnící frekvenci (osa y) a amplitudu (barva dílčích částí), druhý [39] zobrazuje frekvenci (osa x) a amplitudu (osa y) v reálném čase.

6.4.1 Stanovení zadání úkolu

Pomocí frameworku implementujte plugin, který bude zobrazovat frekvenci a amplitudu audio stop. Využijte tutoriály frameworku s rychlou Fourierovou transformací.

Výsledek vyzkoušejte v Audio Plugin Hostu, kde si zapojíte jednu vizualizaci před filtr a druhou za. Vyzkoušejte všechny typy filtru a sledujte rozdíly.

6.4.2 Řešení

Opět si založíme projekt typu audio plugin. JUCE nabízí třídu FFT v modulu DSP. Ten je potřeba přidat v Projuceru:

Modules –> + –> add a module –> global JUCE modules path –> juce_dsp.

Jako první si ve veřejné (public) části PluginProcessoru.h deklarujeme velikosti potřebné pro práci s třídou FFT.

```
enum
{
   fftOrder = 10,
   fftSize = 1 << fftOrder,
   scopeSize = 512
};
```

fftOrder předáme později do konstruktoru třídy FFT, z ní si pomocí operandu << vytvoříme binární reprezentaci čísla 2 ^ 10 (2^fftOrder). Ten nám bude udávat množství vzorků pro transformaci. Hodnota scopeSize bude udávat velikost pole

Dále využijeme obrázek (Image spectogramImage), ve kterém budeme překreslovat jednotlivé pixely spektogramu, proměnnou nextFFTBlockReady typu boolean pro indikaci plného naplnění pole s daty pro transformaci, kterou nastavíme na false, a pole dat scopeData pro vykreslení spektrálního analyzátoru o velikosti scopeSize.

```
Image spectrogramImage;
bool nextFFTBlockReady = false;
float scopeData[scopeSize];
```

V části private si dále deklarujeme objekt forwardFFT třídy dsp::FFT, poté objekt window třídy dsp::WindowingFunction [40] typu float, se kterým bude pracovat analyzér, a pole o velikosti FFT, které pojmenujeme fifo, jelikož s ním budeme pracovat jako s frontou. Poslední dvě deklarace jsou pole pro transformaci a index, pomocí kterého budeme pole

naplňovat. Proto, aby bylo možné provést výpočet FFT, musí data naplňovat pouze první polovinu předaného pole, z toho důvodu je velikost pole s daty dvakrát větší než velikost zásobníku.

```
dsp::FFT forwardFFT;
dsp::WindowingFunction<float> window;

float fifo[fftSize];
float fftData[2 * fftSize];
int fifoIndex = 0;
```

V konstruktoru v PluginProcessor.cpp si inicializujeme forwardFFT tím, že tomuto objektu předáme řád pro vytvoření fourierovi transformace. Obrázku nastavíme PixelFormat na Image::RGB, velikost obrázku na 512x512 a poslední parametr funkce nastavíme na true, ten obrázek vybarví černou barvou (při nastavení na false si obrázek musíte na začátku „vyčistit“ sami). Objektu typu WindowingFunction předáme velikost rozsahu FFT a vybereme jednu z použitelných metod – zde metoda hann.

```
forwardFFT(fftOrder),
spectrogramImage(Image::RGB, 512, 512, true),
window(fftSize, dsp::WindowingFunction<float>::hann)
```

Funkci processBlock složíme ze dvou funkcí z prvního tutoriálu – getNextAudioBlock a pushNextSampleToFifo. Využijeme vyčištění bufferu, které vygenerovala šablona a poté přetypujeme buffer na AudioSourceChannelInfo tím, že jej předáme do konstruktoru objektu bufferToFill.

```
AudioSourceChannelInfo bufferToFill(buffer);
```

Jelikož se chystáme číst data z paměti, nejdříve si ověříme, jestli je vůbec z čeho číst (počet kanálů bufferu je větší než 0). Pokud ano, funkcí getReaderPointer si získáme ukazatel na začátek pole s daty a v cyklu si naplníme frontu hodnotami z bufferu. V moment, kdy máme naplněn počet prvků shodný s velikostí FFT, vyčistíme si zásobník pomocí zeromem a překopírujeme z něj uložená data. Poté zavoláme funkci calculate, která má za úkol provést výpočty a překreslit spektrogram a analyzér.

```
if (bufferToFill.buffer->getNumChannels() > 0)
{
   auto* channelData = bufferToFill.buffer->getReadPointer(0, bufferToFill.startSample);

   for (auto i = 0; i < bufferToFill.numSamples; ++i) {
      if (fifoIndex == fftSize)
      {
         if (!nextFFTBlockReady)
         {
            zeromem(fftData, sizeof(fftData));
            memcpy(fftData, fifo, sizeof(fifo));
            nextFFTBlockReady = true;
            calculate();
         }
         fifoIndex = 0;
      }
      fifo[fifoIndex++] = channelData[i];
   }
}
```

Ve funkci calculate si nejdříve zjistíme šířku a výšku spektogramu – od šířky odečteme 1. Budeme posouvat jednotlivé pixely obrázku, ale při každém posunutí 1 sloupec odstraníme – proto odčítáme jedničku.

```
auto rightHandEdge = spectrogramImage.getWidth() - 1;
auto imageHeight = spectrogramImage.getHeight();
```

Kvůli potlačení prosakování ve spektru aplikujeme funkci pomocí multiplyWithWindowingTable, kde zadáme pole, které chceme modifikovat a velikost dat, která modifikujeme. Poté posuneme všechny, krom krajních sloupců spektrogramu, o jeden sloupec doleva pomocí funkce moveImageSection a provedeme výpočet Fourierovy transformace s daty z bufferu funkcí performFrequencyOnlyForwardTransform.

```
window.multiplyWithWindowingTable(fftData, fftSize);
spectrogramImage.moveImageSection(0, 0, 1, 0, rightHandEdge, imageHeight);
forwardFFT.performFrequencyOnlyForwardTransform(fftData);
```

Dále si zjistíme rozsah hodnot vyprodukovaných Fourierovou transformací pomocí funkce findMinAndMax a postupně v cyklu přepočítáme hodnoty na logaritmickou stupnici, čímž je přichystáme k vizualizaci. Každý pixel pravého krajního sloupce spektogramu vybarvíme barvou podle počtu vzorků vyskytujících se ve spektru.

```
auto maxLevel = FloatVectorOperations::findMinAndMax(fftData, fftSize / 2);


for (auto y = 1; y < imageHeight; ++y)
{
   auto skewedProportionY = 1.0f - std::exp(std::log(y / (float)imageHeight) * 0.2f);
   auto fftDataIndex = jlimit(0, fftSize / 2, (int)(skewedProportionY * fftSize / 2));
   auto level = jmap(fftData[fftDataIndex], 0.0f, jmax(maxLevel.getEnd(), 1e-5f), 0.0f,
1.0f);


   spectrogramImage.setPixelAt(rightHandEdge, y, Colour::fromHSV(level, 1.0f, level,
1.0f));
```

Poté si přepočítáme úroveň pro druhé zobrazení a pro každou hodnotu y si uložíme požadovanou úroveň do pole scopeData.

```
   level = jmap(jlimit(-100.0f, 0.0f, Decibels::gainToDecibels(
         fftData[fftDataIndex])- Decibels::gainToDecibels((float)fftSize)),
         -100.0f, 0.0f, 0.0f, 1.0f);
   scopeData[y] = level;
}
```

Dále potřebujeme přidat přepínání mezi oběma zobrazeními, takže si do souboru PluginEditor.h přidáme Combobox option, kterému přidáme 2 možnosti v konstruktoru (v souboru PluginEditor.cpp) pomocí:

```
option.addItemList(StringArray{ "Spectogram","Visualiser" },1);
option.setSelectedId(1);
```

A tím umožníme výběr mezi spektrogramem a vizualizérem.

Ve funkci resized nastavíme, kde se má ComboBox zobrazit: option.setBounds(10, 10, 100, 30);

Výsledky je potřeba pravidelně překreslovat, takže je potřeba přidat Timer. V hlavičkovém souboru za public AudioProcessorEditor přidáme ", Timer" a v konstruktoru této třídy startTimerHz(30)

Dále přidáme funkci, která se nám zavolá při každém výpočtu Fourierovi transformace a podle ní se bude překreslovat obrázek.

```
void FftAudioProcessorEditor::timerCallback()
{
   if (processor.nextFFTBlockReady) {
      processor.nextFFTBlockReady = false;
      repaint();
   }
}
```

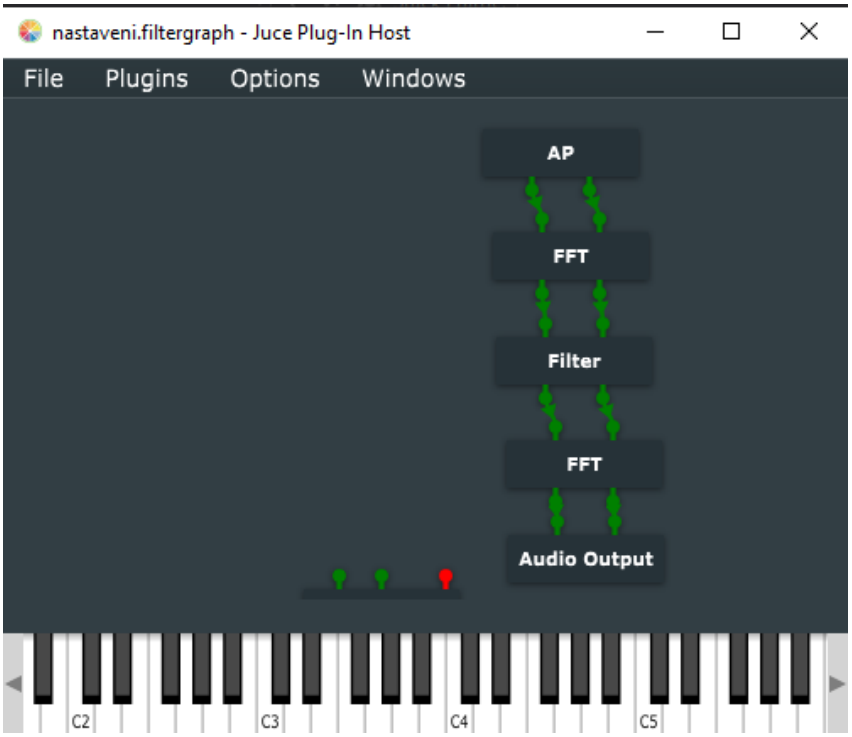
Pak už pouze zbývá doplnit vykreslování ve funkci paint podle vybraného zobrazení. Spektogram máme uložený jako obrázek v PluginProcessoru, tudíž jej stačí pouze vykreslit. U vizualizéru projdeme úrovně jednotlivých frekvencí a spojíme je funkcí g.drawLine, kde amplitudy jednotlivých frekvencí zmapujeme na velikost okna aplikace.

```
g.fillAll(Colours::white);
if(option.getSelectedId()==1)
   g.drawImage(processor.spectrogramImage, getLocalBounds().toFloat());
else {
   g.setColour(Colours::darkred);
   for (int i = 1; i < processor.scopeSize; ++i)
   {
      auto width = getLocalBounds().getWidth();
      auto height = getLocalBounds().getHeight();

      g.drawLine({ (float)jmap(i - 1, 0, processor.scopeSize - 1, 0, width),
            jmap(processor.scopeData[i - 1], 0.0f, 1.0f, (float)height, 0.0f),
               (float)jmap(i,     0, processor.scopeSize - 1, 0, width),
            jmap(processor.scopeData[i],     0.0f, 1.0f, (float)height, 0.0f)
      });
   }
}
```

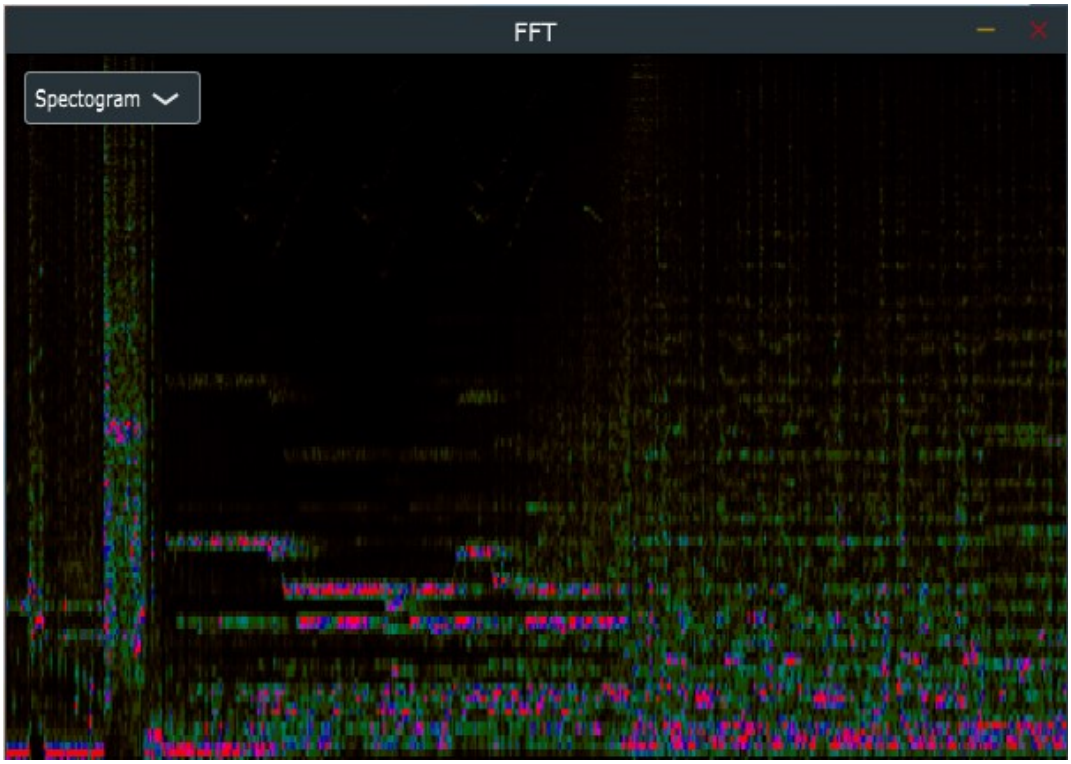
6.4.3 Spojení úkolů v Audio Plugin Hostu

Nyní, když máme hotové 3 různé pluginy, můžeme si je vyzkoušet. Spustíme si Audio Plugin Hosta, přidáme pluginy (viz. kapitola 5.3.1) a spojíme je jako na obrázku níže.



Obrázek 13: Propojení vytvořených pluginů v Audio Plugin Hostu

Spustíme si libovolný audio soubor, filtr nastavíme na horní propust, posuvník pro ořez frekvence na 14000 Hz a vyzkoušíme funkčnost filtru. Ten ze spektra odstranil nízké frekvence:



Obrázek 14: Spektrum před úpravou filtrem

Obrázek 15: Spektrum po úpravě filtrem

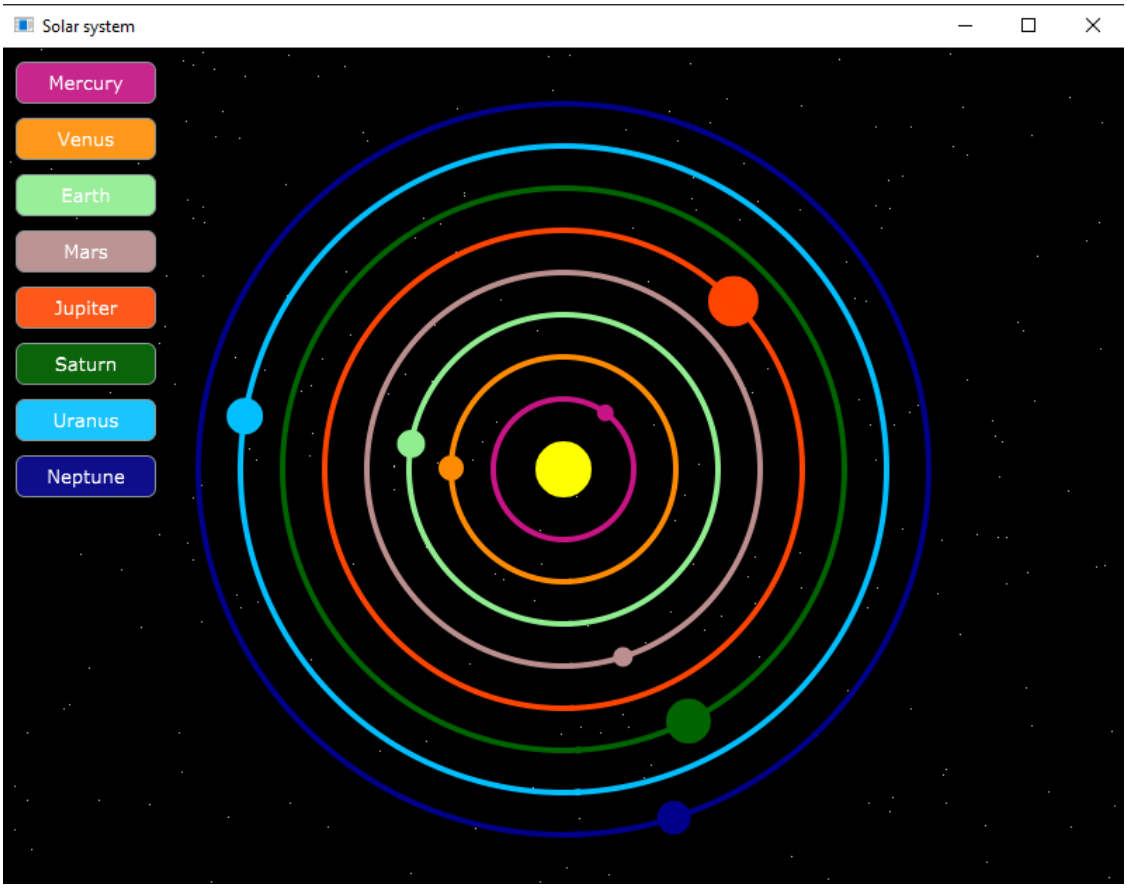
Rozdíl je možné samozřejmě vidět v reálném čase i ve vizualizéru.

6.5 Úkol č. 4 – Animace v JUCE

Poslední úkol bude poněkud odlišný než předešlé. Framework umožňuje tvorbu jednoduchých animací jak pomocí OpenGL, tak díky některým jeho funkcím. Aplikace tohoto typu je vhodná pro tvorbu interaktivního UI. Tato úloha se zaměří zejména na využití funkcí frameworku.

6.5.1 Stanovení zadání úkolu

Pomocí daných struktur vytvořte jednoduchý model Sluneční soustavy podobný obrázku níže.



Obrázek 16: Animovaná aplikace

Podstatou úkolu je rozpohybování jednotlivých planet po kružnicové dráze. Při spuštění aplikace se nejprve vykreslí jednotlivé planety a postupným pohybem se vykreslí i jejich oběžné dráhy. Při změně velikosti okna se dráhy jednotlivých planet smažou a začnou se vykreslovat znovu. Slunce se bude vykreslovat vždy uprostřed okna a jednotlivé dráhy a planety se budou točit kolem něj. Do aplikace přidejte také možnost zobrazení popisu jednotlivých planet – například po kliknutí na tlačítko, vybrání z popupmenu apod. Do pozadí vytvořte efekt blikajících hvězd (žlutých bodů na pozadí).

6.5.2 Řešení

Pro účel tohoto úkolu si necháme vygenerovat šablonu pro animovanou aplikaci (viz kapitola 5.1).

6.5.3 Struktury

V souboru MainComponent.h si vytvoříme 2 struktury – pro hvězdy a planety. Hvězda bude mít 2 členské proměnné, souřadnice třídy Point a integer pro počet jejich zobrazení.

```
struct Star {
   Point<float> coordinates;
   int currentLifeTime;
};
```

Pro planetu budeme potřebovat o něco rozsáhlejší strukturu. Bude obsahovat 3 hodnoty typu float – odsazení od středu (offset), šířku vykreslované planety (size) a rychlost pohybu planety (speed). Dále bod typu float, který značí aktuální polohu pro vykreslení. Path, pomocí které budeme vytvářet kružnici oběžné dráhy planety, colour pro barvu vykreslení, ukazatel na informace k planetě a TextButton, který je v aplikaci zobrazí.

```
struct Planet {
   float offset;
   float size;
   float speed;
   Point<float> actualPosition;
   Path path;
   Colour colour;
   String* info;
   TextButton textButton;
};
```

6.5.4 Deklarace proměnných

Pro čitelnější zápis si definujeme makra COUNT na číslo 8, které nám bude udávat počet vykreslovaných planet, STARS na 256 nebo jiný počet hvězd, který budeme chtít vykreslovat a MAX_LIFETIME na 180, které bude indikovat maximální dobu po kterou se má hvězda zobrazit, než změníme její polohu.

```
#define PLANETS 8
#define STARS 256
#define MAX_LIFETIME 180
```

Dále si vytvoříme pole hvězd, proměnnou typu boolean pro indikaci přidání nové cesty, bod, který bude značit střed okna aplikace, pole planet a druhé pole s informacemi pro zobrazení. Jako poslední si ještě vytvoříme Label, který nám bude zobrazovat popis jednotlivých planet a proměnnou typu int, která bude indikovat, které informace jsou aktuálně zobrazovány.

```
Star stars[STARS];
bool init;
Point<float> center;

Planet planets[PLANETS];
String planetInfo[PLANETS];

Label label;
int currentLabel;
```

Než se přesuneme na tvorbu logiky aplikace, zbývá nám ještě definovat funkce, pro generování hvězd a zobrazování textu v labelu.

```
void showText(int planetID);
void generateStars(int start, int end, bool isInitial);
```

Funkci showText předáme pouze ID planety, pro kterou se mají informace zobrazit. Druhou funkci budeme využívat jak na začátku, pro vygenerování všech hvězd, tak později po vypršení doby, po kterou má být hvězda zobrazena. Budeme jí tedy předávat počáteční a koncový index pole hvězd a proměnnou typu boolean jejíž funkce bude vysvětlena při implementaci funkce.

Nyní máme všechny potřebné struktury, proměnné a funkce nachystané a přesuneme se na logiku aplikace do souboru MainComponent.cpp.

6.5.5 Konstruktor, naplnění struktur

Za konstruktor přidáme dvojtečku a doplníme o planetInfo{“text1“,“text2“…“text8“} a tím naplníme pole planetInfo hodnotami, při použití národní abecedy je potřeba použít CharPointer_UTF8(“text“), jinak by program padal. V konstruktoru díky šabloně již voláme funkci setSize. Hodnoty změníme na 800,600 a tím nastavíme velikost okna aplikace. Dále je zde volaná funkce setFramesPerSecond, která jak již název napovídá udává počet snímků za sekundu, její hodnotu necháme na 60.

Dále si deklarujeme několik konstant. Pro barvy, odsazení planet, jejich velikost a rychlost, kterou se mají pohybovat, a pole řetězců s názvem jednotlivých planet.

```
const Colour colours[PLANETS] = {
    Colours::mediumvioletred, Colours::darkorange,
    Colours::lightgreen, Colours::rosybrown,
    Colours::orangered, Colours::darkgreen,
    Colours::deepskyblue, Colours::darkblue
};

const float offset = 50.0f;
const float planetSizes[PLANETS] = { 12.0f,18.0f,20.0f,14.0f,36.0f,32.0f,26.0f,24.0f };
const float speeds[PLANETS] = { 0.04f,0.05f,0.035f,0.03f,0.025f,0.045,0.035f,0.03f };

const String names[] = { "Mercury","Venus","Earth","Mars","Jupiter"
      ,"Saturn","Uranus","Neptune" };
```

Poté si v cyklu projdeme pole s planetami a naplníme je příslušnými hodnotami. Tlačítkům nastavíme příslušnou barvu pomocí setColour, text zobrazení pomocí setButtonText a přiřadíme jim anonymní funkci pro event onClick, která zobrazí příslušné informace o planetě. Jako poslední je samozřejmě potřeba tlačítko zviditelnit pomocí funkce addAndMakeVisible.

```
for (int i = 0; i < PLANETS; i++) {
    planets[i].colour = colours[i];
    planets[i].offset = offset + i * 30;
    planets[i].size = planetSizes[i];
    planets[i].speed = speeds[i];
    planets[i].info = &planetInfo[i];
    planets[i].textButton.setColour(
        TextButton::ColourIds::buttonColourId
        ,planets[i].colour
    );
    planets[i].textButton.setButtonText(names[i]);
    planets[i].textButton.onClick = [this,i] {showText(i); };
    addAndMakeVisible(&planets[i].textButton);
}
```

Zbývá nastavit popisek. Dáme mu malou průhlednost pomocí funkce setAlpha, nastavíme font písma pomocí setFont a barvu funkcí setColour. Poté ho přidáme do aplikace a zneviditelníme jej pomocí setVisible(false).

```
label.setAlpha(0.92f);
label.setFont(20.0f);
label.setColour(Label::backgroundColourId, Colours::white);
addAndMakeVisible(&label);
label.setVisible(false);
```

6.5.6 Pomocné funkce

Funkce pro zobrazení informací o jednotlivých planetách implementujeme tak, aby se při kliknutí na tlačítko zobrazil příslušný text, po kliknutí na jiné tlačítko se pouze upraví text a změní barva textu a po opětovném kliknutí na stejné tlačítko label zmizí.

```
void MainComponent::showText(int planetID) {
    bool isVisible = label.isVisible();
    if (isVisible && currentLabel == planetID) {
        label.setVisible(false);
    }
    else {
        label.setVisible(true);
        currentLabel = planetID;
        label.setColour(Label::textColourId, planets[planetID].colour);
        label.setText(*planets[planetID].info, NotificationType::dontSendNotification);
    }
}
```

Generování hvězd zajistíme podle proměnné isInitial. Pokud je volána s hodnotou true, vygenerujeme hvězdám náhodnou polohu a hodnotu proměnné currentLifeTime, čímž zajistíme, že se nám nebudou měnit všechny zároveň po určité době, ale budou měnit polohu postupně. To je žádoucí, generujeme-li hvězdy poprvé. Při volání s hodnotou false pouze vynulujeme currentLifeTime a změníme polohu hvězdy.

```
void MainComponent::generateStars(int start, int end, bool isInitial) {


    for (int i = start; i < end; i++) {
        int randomWidth = Random::getSystemRandom().nextInt(Range<int>(1,
getWidth()));
        int randomHeight = Random::getSystemRandom().nextInt(Range<int>(1,
getHeight()));
```

```
        if (isInitial) {
            stars[i].currentLifeTime = Random::getSystemRandom().nextInt(
                Range<int>(1, MAX_LIFETIME));
        }
        else stars[i].currentLifeTime = 0;

        stars[i].coordinates.setXY(randomWidth, randomHeight);
    }
}
```

6.5.7 Funkce resized

Zde zajistíme, aby se nám při změně velikosti okna přepočítala pozice středu (Slunce), určíme minimální velikost okna a nastavíme pozici, kde se mají vykreslit tlačítka a popisek s informacemi o planetách.

Nejdříve si zjistíme výšku a šířku okna, ty si uložíme do proměnných x a y. Pokud budou nižší než námi požadovaná minimální velikost, tak hodnotu nastavíme na minimální požadovanou a poté změníme velikost okna pomocí setSize. Střed aktualizujeme pomocí setXY na polovinu výšky a šířky.

```
float x = getWidth();
float y = getHeight();

if (x < 600)
    x = 600;

if (y < 500)
    y = 500;

setSize(x, y);
center.setXY(x / 2, y / 2);
```

Zvolíme si odsazení shora a zleva, šířku a výšku tlačítek, a v cyklu si nastavíme jejich velikost funkcí setBounds. Také smažeme oběžnou dráhu planety funkcí clear. K odsazení shora přičteme 40, 30 šířku tlačítka a 10 odsazení mezi tlačítky. Proměnnou init typu boolean nastavíme na true, čímž dáme najevo, že se mají oběžné dráhy planet postupně zaznamenat a vykreslovat znovu. Jako poslední nastavíme label s texty k jednotlivým planetám do spodní části aplikace s odsazením 10 od krajů.

```
const int fromLeft = 10;
int fromTop = 10;
const int width = 100;
const int height = 30;

for (int i = 0; i < PLANETS; i++) {
   planets[i].path.clear();
   planets[i].textButton.setBounds(fromLeft, fromTop, width, height);
   fromTop += 40;
}
init = true;
label.setBounds(10, 4 * getHeight() / 5, getWidth() - 20, getHeight() / 5 - 10);
```

6.5.8 Funkce paint

Jak již bylo zmíněno, funkce paint je volána pro vykreslení jednotlivých snímků. My budeme využívat zejména funkci fillEllipse(float souřadnice X, float souřadnice Y, float šířka, float výška), která vykreslí elipsu (při shodné výšce a šířce kruh) na zadaných souřadnicích.

Nejprve ale pozadí vyplníme černou barvou. Poté nastavíme barvu na světle žlutou a v cyklu vykreslíme hvězdy. Nejdříve ale zkontrolujeme jejich „životnost“ a pokud již dosáhla maximální doby zobrazení, tak ji necháme funkcí generateStars vygenerovat novou polohu a nastavit currentLifeTime na 0. Po vykleslení hvězd ještě vykreslíme uprostřed okna Slunce žlutou barvou. Polovinu jeho šířky odečteme od obou souřadnic, aby bylo opravdu vprostřed okna a nezačínalo kousek dál.

```
g.fillAll(Colours::black);

g.setColour(Colours::lightyellow);

for (int i = 0; i < STARS; i++) {
```

```
    if (stars[i].currentLifeTime > MAX_LIFETIME)
        generateStars(i, i + 1, false);

    g.fillEllipse(stars[i].coordinates.x, stars[i].coordinates.y, 1.0f, 1.0f);
}

g.setColour(Colours::yellow);
g.fillEllipse(center.x - 20.0f, center.y - 20.0f, 40.0f, 40.0f);
```

Poté vykreslíme planety. V cyklu nejdříve nastavíme barvu pro vykreslení a poté podle aktuální pozice planety a její velikosti vykreslíme jí samotnou. Jako poslední zvýrazníme oběžnou dráhu planety čárou o velikosti 4 pixely.

```
for (int i = 0; i < PLANETS; i++) {
    g.setColour(planets[i].colour);
    g.fillEllipse(planets[i].actualPosition.x - planets[i].size / 2,
        planets[i].actualPosition.y - planets[i].size / 2,
        planets[i].size,
        planets[i].size
    );
    g.strokePath(planets[i].path, PathStrokeType(4.0f));
}
```

6.5.9 Funkce update

Aby se nám planety a hvězdy nevykreslovali stále na jednom místě, je potřeba přepočítávat jejich polohu. K tomu je vhodná právě funkce update, kterou máme připravenou díky šabloně aplikace.

Nejdříve všem hvězdám zvýšíme životnost o 1. V jednom z tutoriálů na stránkách frameworku můžeme najít využití počítadla snímků (funkce getFrameCounter) pro animování dráhy. Pokud použitý vzorec lehce upravíme, můžeme jednoduše vykreslit kruh. [41]

Po změně polohy planety rozhodneme, co se má dít s oběžnou dráhou. Pokud proměnná init má hodnotu true, založíme novou cestu, pokud nemáme ještě vykreslenou celou kružnici, tak přidáváme další body do cesty a až je již obkreslena celá, tak cestu uzavřeme.

```
void MainComponent::update()
{
for (int i = 0; i < STARS; i++)
   stars[i].currentLifeTime++;

for (int j = 0; j < PLANETS; j++) {
   planets[j].actualPosition.setXY(
      center.x + planets[j].offset * std::sin(getFrameCounter() * planets[j].speed),
      center.y + planets[j].offset * std::cos(getFrameCounter() * planets[j].speed)
   );

   if (init) {
      planets[j].path.startNewSubPath(planets[j].actualPosition);
   }
   else if (planets[j].path.getLength() < (MathConstants<float>::twoPi* planets[j].offset))
      planets[j].path.lineTo(planets[j].actualPosition);
   else
      planets[j].path.closeSubPath();
   }
init = false;
}
```

ZÁVĚR

Práce se snaží seznámit čtenáře s frameworkem JUCE a možnostmi jeho využití ve výuce programování za pomocí jazyka C++ a vývojových prostředí Projucer a Visual Studio. Cílem bylo uvést čtenáře do práce s frameworkem a poté navrhnout několik dílčích úloh, které by měly zejména prakticky demonstrovat znalosti získané při studiu za předpokladu alespoň základní znalosti programování v jazyce C++. Byly navrženy úkoly pro studenty, pomocí kterých by si měli vyzkoušet práci s frameworkem za využití výborného nástroje – Audio Plugin Host.

Z hlediska dalšího možného využití frameworku by se dalo, jako například téma jiné bakalářské práce, vytvořit syntetizér, třeba i s využitím MIDI příslušenství, přece jenom k tomu je framework primárně určen.

SEZNAM POUŽITÉ LITERATURY

[1] *ROLI* [online]. [cit. 2020-02-18]. Dostupné z: https://roli.com/

[2] *What is MIDI (Musical Instrument Digital Interface)* [online]. [cit. 2020-02-18]. Dostupné z: https://whatis.techtarget.com/definition/MIDI-Musical-Instrument-Digital-Interface

[3] Projucer. *JUCE* [online]. [cit. 2020-02-18]. Dostupné z: https://juce.com/discover/projucer

[4] *Tutorials | JUCE* [online]. [cit. 2020-02-18]. Dostupné z: https://juce.com/learn/tutorials

[5] DOCUMENTATION |JUCE. JUCE [online]. [cit. 2020-03-09]. Dostupné z: https://juce.com/learn/documentation

[6] ROBINSON, Martin. *Getting Started with JUCE*. 1. Birmingham: Packt Publishing, 2013. ISBN 978-1783283319.

[7] The Audio Programmer - YouTube. *YouTube* [online]. [cit. 2020-03-09]. Dostupné z: https://www.youtube.com/channel/UCpKb02FsH4WH4X_2xhIoJ1A

[8] JUCE - JUCE Forum. *JUCE* [online]. [cit. 2020-03-09]. Dostupné z: https://forum.juce.com/

[9] STROUSTRUP, Bjarne. *The C++ programming language*. Fourth edition. Upper Saddle River, NJ: Addison-Wesley, [2013]. ISBN 978-0321563842.

[10] ISO/ IEC JTC1/SC22/WG21 - The C++ Standards Committee - ISOCPP. *Open Standards* [online]. [cit. 2020-03-31]. Dostupné z: http://www.open-std.org/jtc1/sc22/wg21/

[11] Visual Studio Code Frequently Asked Questions. *Visual Studio Code* [online]. [cit. 2020-02-18]. Dostupné z: https://code.visualstudio.com/docs/supporting/FAQ

[12] The GNU Operating System and the Free Software Movement. *O projektu GNU – Projekt GNU – Nadace pro svobodný software* [online]. 2016 [cit. 2020-06-11]. Dostupné z: http://www.gnu.org/

[13] About the Eclipse Foundation | the Eclipse Foundation. *Enabling Open Innovation & Collaboration | The Eclipse Foundation* [online]. [cit. 2020-03-31]. Dostupné z: https://www.eclipse.org/org/

[14] Qt Creator. *Qt Wiki* [online]. [cit. 2020-03-31]. Dostupné z: https://wiki.qt.io/Qt_Creator

[15] The Qt Governance Model. *Qt Wiki* [online]. [cit. 2020-03-31]. Dostupné z: https://wiki.qt.io/The_Qt_Governance_Model

[16] *JetBrains: Developer Tools for Professionals and Teams* [online]. [cit. 2020-03-31]. Dostupné z: https://www.jetbrains.com/

[17] Lambda Expressions in C++ | Microsoft Docs. *Microsoft | Docs* [online]. [cit. 2020-03-02]. Dostupné z: https://docs.microsoft.com/en-us/cpp/cpp/lambda-expressions-in-cpp?view=vs-2019

[18] Enumeration declaration. *Cppreference.com/* [online]. [cit. 2020-03-31]. Dostupné z: https://en.cppreference.com/w/cpp/language/enum

[19] Data structures - C++ Tutorials. *Cplusplus.com - The C++ Resources Network* [online]. [cit. 2020-03-31]. Dostupné z: http://www.cplusplus.com/doc/tutorial/structures/

[20] GREGOIRE, Marc. *Professional C++*. 4. Wrox, 2018. ISBN 978-1119421306.

[21] JUCE: FileChooser Class Reference. *JUCE* [online]. [cit. 2020-03-02]. Dostupné z: https://docs.juce.com/master/classFileChooser.html

[22] JUCE | AudioFormatReader Class Reference. *JUCE* [online]. [cit. 2020-03-02]. Dostupné z: https://docs.juce.com/master/classAudioFormatReader.html

[23] JUCE: AudioFormatManager Class Reference. *JUCE* [online]. [cit. 2020-03-02]. Dostupné z: https://docs.juce.com/master/classAudioFormatManager.html

[24] JUCE: AudioFormatReaderSource Class Reference. *JUCE* [online]. [cit. 2020-03-02]. Dostupné z: https://docs.juce.com/master/classAudioFormatReaderSource.html

[25] JUCE: JUCE: AudioTransportSource Class Reference. *JUCE* [online]. [cit. 2020-03-02]. Dostupné z: https://docs.juce.com/master/classAudioTransportSource.html

[26] JUCE: Slider Class Reference. *JUCE* [online]. [cit. 2020-03-09]. Dostupné z: https://docs.juce.com/master/classSlider.html#details

[27] JUCE: ComboBox Class Reference. *JUCE* [online]. [cit. 2020-03-09]. Dostupné z: https://docs.juce.com/master/classComboBox.html#details

[28] JUCE | Button Class Reference. *JUCE* [online]. [cit. 2020-03-24]. Dostupné z: https://docs.juce.com/master/classButton.html

[29] JUCE | Point< ValueType > Class Template Reference. *JUCE* [online]. [cit. 2020-03-24]. Dostupné z: https://docs.juce.com/master/classPoint.html

[30] Path Class Reference. *JUCE* [online]. [cit. 2020-03-24]. Dostupné z: https://docs.juce.com/master/classPath.html

[31] JUCE: maths. *JUCE* [online]. [cit. 2020-03-31]. Dostupné z: https://docs.juce.com/master/group__juce__core-maths.html

[32] Klasifikace filtrů podle impulsní charakteristiky (FIR, IIR). Matematická biologie [online]. [cit. 2020-07-13]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-modelovani-dynamickych-biologickych-dat--linearni-a-adaptivni-zpracovani-dat--linearni-filtrace-filtry-1--klasifikace-filtru-podle-impulsni-charakteristiky-fir-iir

[33] LYONS, Richard G. *Understanding Digital Signal Processing: Unders Digital Signal Proces*. 3. 501 Boylston Street, Suite 900 Boston, MA 02116: Prentice Hall, 2010. ISBN 0137027419.

[34] AudioProcessor Class Reference. *JUCE: Class Index* [online]. [cit. 2020-02-18]. Dostupné z: https://docs.juce.com/master/classAudioProcessor.html#abbac77f68ba047cf60c4bc97326dcb58

[35] Tutorial: Build an audio player. *Tutorials|JUCE* [online]. [cit. 2020-02-18]. Dostupné z: https://docs.juce.com/master/tutorial_playing_sound_files.html

[36] JUCE: AudioSource Class Reference. *Docs.juce.com* [online]. [cit. 2020-03-05]. Dostupné z: https://docs.juce.com/master/classAudioSource.html#a653279dbd167f70e318fd280681dbddf

[37] JUCE: AudioVisualiserComponent Class Reference. *JUCE* [online]. [cit. 2020-03-25]. Dostupné z: https://docs.juce.com/develop/classAudioVisualiserComponent.html

[38] JUCE: Tutorial: The fast Fourier transform. *JUCE* [online]. [cit. 2020-03-25]. Dostupné z: https://docs.juce.com/master/tutorial_simple_fft.html

[39] JUCE: Tutorial: Visualise the frequencies of a signal in real time. *JUCE* [online]. [cit. 2020-03-25]. Dostupné z: https://docs.juce.com/master/tutorial_spectrum_analyser.html

[40] JUCE: dsp::WindowingFunction< FloatType > Class Template Reference. *JUCE* [online]. [cit. 2020-03-25]. Dostupné z: https://docs.juce.com/master/classdsp_1_1WindowingFunction.html#a3e8b398d754d06f6b6863639d0de52cca0601ab384b75f103d84d1c5e5dd22c20

[41] JUCE: Tutorial: Animating geometry. *Docs.juce.com* [online]. [cit. 2020-03-06]. Dostupné z: https://docs.juce.com/master/tutorial_animation.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MIDI Musical instrumental digital interface

UI User interface

GUI Graphical user interface

DSP Digital signal processing

IIR Infinite impulse response

DFT Discrete Fourier transform

FFT Fast Fourier transform

IDE Integrated development Environment

SEZNAM OBRÁZKŮ

Obrázek 1: Slider 17
Obrázek 2: ComboBox 18
Obrázek 3: TextButton 18
Obrázek 4: Projucer 21
Obrázek 5: Založení nového projektu 1 22
Obrázek 6: Založení nového projektu 2 22
Obrázek 7: doplnění cest k dostupným pluginům 24
Obrázek 8: Zjištění dostupnosti pluginů podle zadaných cest 25
Obrázek 9: Okno s cestami pro vyhledání pluginů 25
Obrázek 10: Propojení v Audio Plugin Hostu 26
Obrázek 11: Audiopřehrávač 38
Obrázek 12: Filtr 43
Obrázek 13: Propojení vytvořených pluginů v Audio Plugin Hostu 49
Obrázek 14: Spektrum před úpravou filtrem 50
Obrázek 15: Spektrum po úpravě filtrem 50
Obrázek 16: Animovaná aplikace 51

SEZNAM PŘÍLOH

Příloha P I: Obsah přiloženého CD-ROM disku

PŘÍLOHA P I: OBSAH PŘILOŽENÉHO CD-ROM DISKU

- Elektronická verze bakalářské práce
- Složka se zdrojovými kódy vypracovaných úkolů