ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Bakalářská práce

Modulární syntéza pomocí softwaru Pure Data ve VR

Ondřej Pelikán
Počítačové hry a grafika

Květen 2024
Vedoucí práce: Ing. Ondřej Slabý

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pelikán** Jméno: **Ondřej** Osobní číslo: **507645**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Modulární syntéza pomocí softwaru Pure Data ve VR

Název bakalářské práce anglicky:

Modular synthesis using Pure Data in VR

Pokyny pro vypracování:

Seznamte se se softwarem Pure Data a jeho architekturou. Soustřeďte se na možnosti spouštění hotových patchů, programových bloků pro zpracování zvuku sestavených ve vizuálním prostředí Pure Data, v externích aplikacích pro generaci a úpravu zvukového signálu. Dále se seznamte s principy návrhu uživatelských rozhraní pro virtuální realitu. Navrhněte uživatelské rozhraní pro virtuální realitu, které umožní vytvářet a propojovat objekty reprezentující hotové bloky vytvořené v softwaru Pure Data intuitivním způsobem. Implementujte navržené rozhraní v enginu vlastního výběru a propojte jej se softwarem Pure Data tak, aby bylo možné pouze použitím vašeho rozhraní a hotových bloků generovat a dále upravovat zvuk. Rozhraní musí podporovat zapojení alespoň deseti bloků. Nastavení aplikace, např. volba výstupního zařízení, může být řešeno mimo prostředí virtuální reality. Zamyslete se nad rozšířením vaší implementace o tvorbu vlastních bloků ze základních stavebních prvků používaných softwarem Pure Data. Za účelem testování funkčnosti aplikace vytvořte pomocí softwaru Pure Data alespoň deset bloků, které bude možné v aplikaci nahrát. Výslednou aplikaci kvalitativně otestujte s alespoň pěti uživateli, zaměřte se kromě zvukových funkcí aplikace na přirozenost propojování a ovládání bloků. Na základě výsledků z testování zvažte a realizujte vhodné úpravy aplikace.

Seznam doporučené literatury:

1) A. Kramarzewski and E. De Nucci. Practical Game Design : Learn the Art of Game Design Through Applicable Skills and Cutting-Edge Insights, Packt Publishing, Limited, 2018.
2) S. Stahlke and P. Mirza-Babaei. The Game Designer's Playbook : An Introduction to Game Interaction Design, Oxford University Press, Incorporated, 2022.
3) W. R. Sherman and A. B. Craig. Understanding virtual reality : interface application and design second edition (Second). Elsevier Science and Technology Books, 2019.
4) J. Jerald. The VR Book: Human-Centered Design for Virtual Reality. Association for Computing Machinery and Morgan & Claypool. 2015.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ondřej Slabý katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **16.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. Ondřej Slabý podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování / Prohlášení

Děkuji především svému vedoucímu inženýru O. Slabému, který neúnavně odpovídal na mé otázky a strávil mnoho dní diskutováním o možnostech a implementacích.

Dále bych chtěl poděkovat Katedře počítačové grafiky a interakce a Fakultě elektrotechnické za příležitost spojit několik svých zájmů v jedné práci.

V neposlední řadě děkuji své rodině za trpělivost a podporu.

Prohlašuji, že jsem tuto práci vypracoval samostatně a že jsem veškeré zdroje uvedl v seznamu literatury.

V Praze, 23. května 2024

.....

Abstrakt / Abstract

Hlavním tématem této práce je integrace Pure Data patchů do virtuální reality pomocí Unreal Enginu a LibPD. Výsledkem je aplikace spustitelná v operačním systému Windows propojeným s platformou Oculus Quest. Uživatel v této aplikaci vytváří modulární syntezátor, ve kterém jednotlivé moduly představují Pure Data patche.

Text nejdříve popisuje relevantní pojmy a analyzuje existující implementace a návrhové směrnice pro vývoj ve virtuální realitě. V návrhu popisuje, že důležitá část této práce je doplnění momentálního trhu existujících implementací o realistické vizuály a interakce. Implementační část podrobně rozebírá veškeré procesy, které vedly k vytvoření iluze práce se skutečným syntezátorem. V závěru vyhodnocuje testování a představuje možné rozšíření práce.

Klíčová slova: virtualní realita, Pure Data, LibPD, Unreal Engine, modulární syntéza

The main subject of this work is the integration of Pure Data into virtual reality using Unreal Engine and LibPD. The product is an application for Windows and Oculus Quest. In this application, the user creates a modular synthesizer, in which modules represent Pure Data patches.

This document first explains relevant concepts and analyzes existing implementations and design practices for virtual reality development. The design section describes that an important part of this work is supplementing the current market of existing implementations with realistic visuals and interactions. The implementation part discusses in detail all the processes that led to the creation of the illusion of working with a real synthesizer. In the end, it evaluates the testing and presents a possible extension of this work.

Keywords: virtual reality, Pure Data, LibPD, Unreal Engine, modular synthesis

Title translation: Modular synthesis using Pure Data in VR

Obsah /

Tabulky /

Kapitola 1
Úvod

Jeden z největších přínosů vývoje virtuální reality za posledních 20 let je zpřístupnění různých zážitků širšímu publiku. To je jedno z hlavních témat mé práce. Představena bude aplikace s modelem modulárního syntezátoru. Skutečné modulární syntezátory jsou elektronické hudební nástroje, které mohou být pro mnoho lidí z ekonomických a dalších důvodů nedosažitelné. Existují desktopové aplikace s cílem imitace těchto nástrojů. Nedosahují však imerzivity potřebné k nahrazení pocitu prostorového skládání a propojování jednotlivých modulů. Naopak právě virtuální realita je perfektním médiem pro tuto úlohu.

Aplikace, která je výsledkem této práce, doplňuje momentální trh o realistické vizuály a interakce. Technická strana také inovuje. Veškeré zpracování zvuku ovládá programovací jazyk Pure Data. Další probíranou problematikou je tedy integrace původně grafického programovacího jazyka Pure Data s herním enginem. Představeny budou různé možnosti řešení a následně i podrobný popis toho nejvhodnějšího s využitím knihovny LibPD.

Autentické vizuály a propojení s virtuální realitou obstarává Unreal Engine. Unreal Engine společně s definovanými návrhovými praktikami pro virtuální realitu zprostředkovává také intuitivní interakce a celkovou přirozenost simulace. Jedním z cílů při tvorbě tohoto textu bylo srozumitelné popsání všech různých procesů potřebných k vytvoření kýžené iluze, že uživatel ovládá skutečný modulární syntezátor.

Vzniklá aplikace prošla testováním a popsány budou její úspěchy a možnosti k rozšíření. Dokument může sloužit hned několika způsoby. Nabízí přehled dosavadních úspěchů v tomto oboru. Zevrubné shrnutí návrhových praktik v analýze umožňuje rychlé představení klíčových výzev při vývoji virtuálních světů. Komplexní popis veškerého kódu se nabízí jako průvodce pro budoucí práce využívající některé ze zahrnutých platforem (Pure Data patche, LibPD, Unreal Engine pluginy, virtuální realita a další). Samotná aplikace může být výchozím bodem pro rozšíření implementovaných funkcionalit o tvorbu nových modulů za běhu aplikace.

Kapitola 2
Analýza

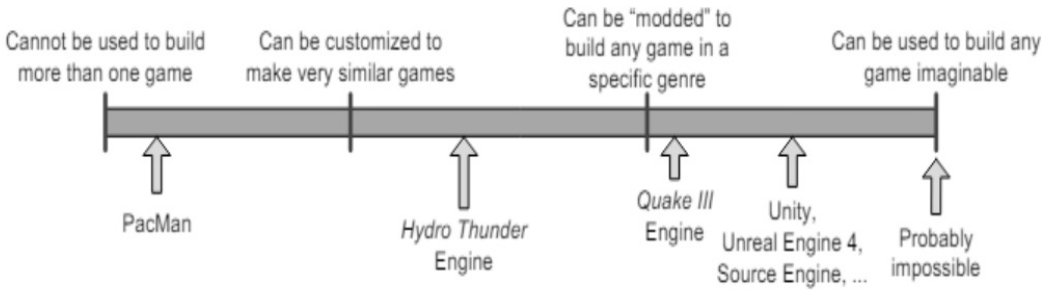
Tato kapitola nejdříve uvede důležité termíny a následně bude definovat kritéria pro hodnocení existujících implementací na téma modulární syntézy ve virtuální realitě. Poté postupně projde zmíněné implementace od nejpopulárnějších. Pokračovat bude shrnutím správných praktik pro návrh uživatelského rozhraní pro aplikace ve virtuální realitě.

2.1 Analýza vývojového prostředí

Za prvé se musíme zabývat představením dílčích programů v projektu. Nejedná se o komplexní popis různých odvětví herního vývoje, ale o představení terminologie.

2.1.1 Herní enginy

Termín herní enginy se poprvé objevuje v polovině 90. let s příchodem her jako Doom a Quake III [1]. Tyto hry implementovaly oddělenou část výpočtu grafiky a jejího obsahu. Definované nástroje se dále používaly v dalších titulech a souhrn takových nástrojů můžeme brát jako herní engine.



Obrázek 2.1. Časový vývoj her směrem k tvorbě enginů [1].

Skutečně univerzální herní engine zatím nebyl vytvořen a možná ani nikdy nebude. Různé enginy jsou vyvíjeny s jasnými omezeními za účelem optimalizace her, které v něm budou tvořeny. Ačkoliv je v moderních enginech (Unity, Unreal Engine nebo Godot) možné vytvářet hry různých žánrů, jsou enginy často spojovány s žánry, pro které si je studia původně vytvořila (Epic games představilo Unreal Engine v roce 1998 pro jejich First-Person-Shooter hru Unreal)[1].

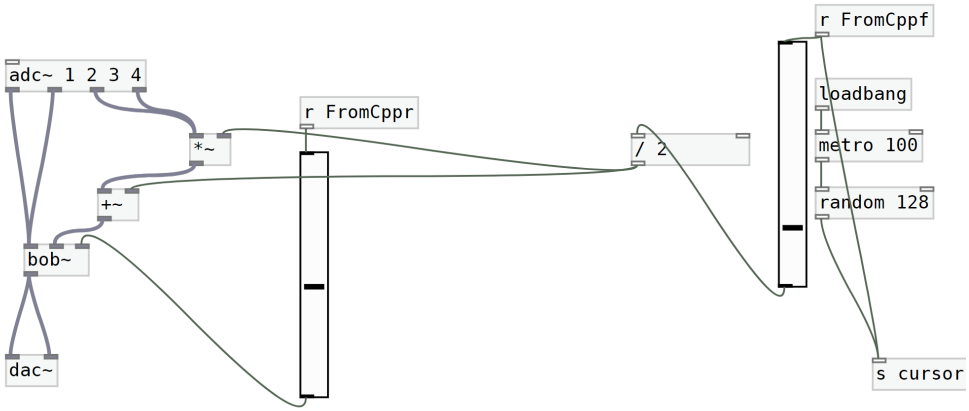
Na závěr uvedu definici herní engine od A. Andrade:

Herní engine je opakovaně použitelná softwarová vrstva umožňující oddělení běžných herních konceptů od herního obsahu (úrovně, grafika atd.) [2].

2.1.2 Pure Data

Pure Data je grafický programovací jazyk vytvořený Millerem S. Puckettem v 90. letech jako nekomerční software adresující různé problémy jazyka Max. Max byl vyvíjen s

myšlenkou programování ve stylu modulárních syntezátorů. Jednotlivé programy se nazývají patche a skládají se z bloků na obrazovce představujících data a funkce. Tyto bloky se mezi sebou propojují a vytváří tzv. DSP - Digital Sound Processing. V praxi takový program dokáže číst a upravovat tok dat, ať už zvukový nebo vizuální. [3]



Obrázek 2.2. Ukázka Pure Data patche z této práce.

Díky intuitivnímu pracovnímu postupu se kolem Pure Data vytvořila veliká komunita aktivních uživatelů, kteří využívají toho, že je jazyk open source a tvoří nové verze a funkcionality (např. síťovou komunikaci). Momentálně existuje mnoho rozšíření tohoto jazyku jako Purr data nebo Plug data a ty se stále vyvíjí. Mezi uživateli jsou jak programátoři, tak tvůrci audiovizuálního umění, protože ve spojení například s platformou Arduino jsou Pure Data vhodným nástrojem pro tvorbu audiovizuálních uměleckých inscenací. Další výhodou pro umělce je fakt, že patche je možné měnit v reálném čase (na rozdíl od tradičních programovacích jazyků) a pomocí toho je možné vytvářet dynamická umělecká představení. [4]

2.1.3 Virtuální realita

The VR Book definuje virtuální realitu jako počítačově generované digitální prostředí, které lze zažít a interagovat s ním, jako by to bylo prostředí skutečné [5]. Uživatelé se s ní už dnes mohou setkat v mnoha různých oborech. Jako první možná přijdou na mysl videohry, ale zařízení augmented reality se využívají i třeba v moderních továrnách jako má Intel [6].

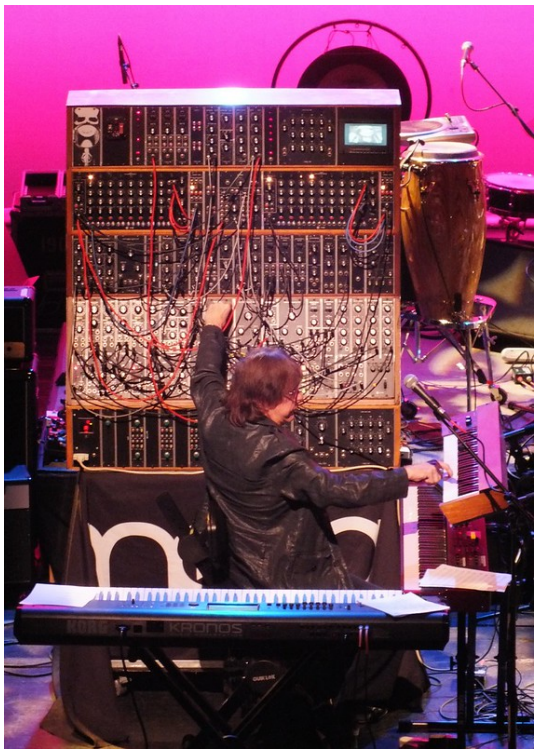
Stále více se mluví o začlenění virtuální reality do výuky. A s nárůstem dostupnosti modelů Oculus Quest vidíme také nárůst nových druhů obsahu dělaných specificky pro jednotlivce s virtuální realitou. Popularitě pomohl v posledním roce především Apple a jeho nový Apple Vision Pro, který přivedl k problematice virtuální reality mnoho nových zájemců (ačkoliv svou technologii jako virtuální realitu nenazývá).

Dostupná virtuální realita otevírá nové možnosti pro výpočetní technologie. Může vytvořit zcela nové druhy interakcí a zpřístupnit zážitky z fyzického světa. Přesně tomu se tato práce věnuje.

2.1.4 Modulární syntezátory

Syntezátor je zařízení vytvářející zvuk definováním jeho výšky, barvy a hlasitosti. První zařízení odpovídající tomuto popisu se objevuje již v roce 1986 od autora Thadseuse Cahilla. V druhé polovině 20. století se s nově vznikajícími hudebními žánry objevují

v mainstreamu legendární vynálezy v tomto oboru od Boba Mooga nebo Laurense Hammonda. [7] Syntezátory mohou vytvářet zvuk mnoha různými způsoby (aditivní a substraktivní syntéza, frekvenční modulace a další).



Obrázek 2.3. Keith Emerson se svým modulárním syntezátorem značky Moog (John Grabowski, 2014).

Modulární syntezátory jsou zařízení definované svou fyzickou strukturou. Namísto uceleného předem sestaveného nástroje jsou složené z jednotlivých modulů. Moduly po propojení tvoří celek produkující zvukový signál. Signál mohou moduly generovat, upravovat ho nebo míchat.

Propojení je realizováno kabely, které muzikant zapojuje do vstupů a výstupů na každém modulu. Díky této praktice jsou to často velká a na první pohled chaotická zařízení. Na druhou stranu má uživatel naprostou kontrolu nad cestou, kterou signál teče a barva finálního zvuku je u různých jednotek velice individuální, protože si muzikant může připojit moduly podle svého zalíbení.

2.2 Existující aplikace

Existující aplikace je důležité zohlednit při tvorbě nové práce z několika důvodů. Je bezúčelné vytvořit aplikaci totožnou s již existující aplikací. Předešlé aplikace nás mohou inspirovat, ale také poukázat na cesty, které nevedou k dobrým výsledkům. Důležité je také zohlednit odezvu na existující aplikace. Z těchto důvodů budu deklarovat několik kritérií a následně hodnotit, které implementace je splňují a které ne. Následně si vyberu, kterými implementacemi bych se chtěl inspirovat a ve kterých bych rád něco změnil.

Kritéria nezohledňují obecnou kvalitu aplikací, ale pouze ty kvality, které jsem si pro svou implementaci stanovil jako důležité vzhledem k cílové skupině definované v 3.1. Více se k tomuto tématu vrátím v kapitole 3.

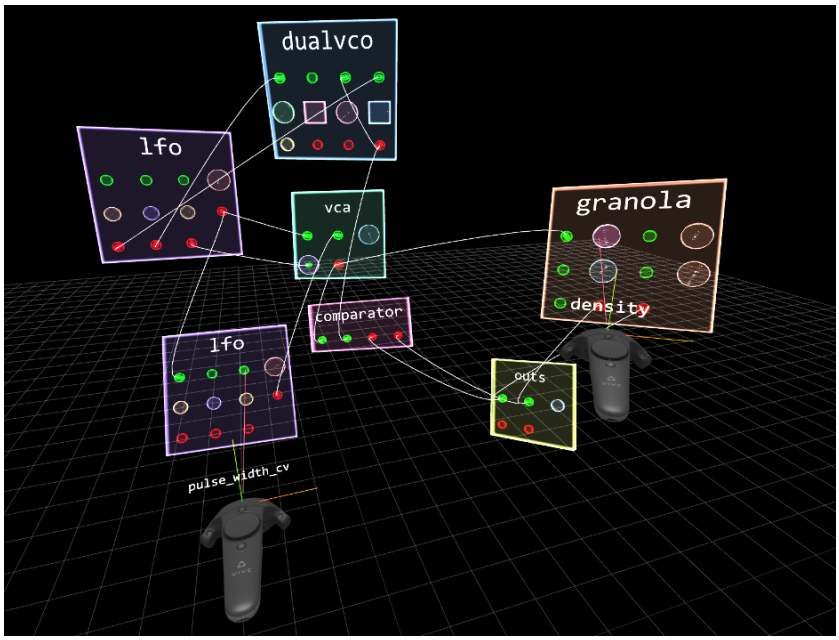
Kritéria:

- Vizuální podobnost virtuálního syntezátoru ke skutečným modulárním syntezátorům
- Modularita systému tvorby zvuku
- Kvalita zvuku
- Možnosti přizpůsobení prostředí
- Vhodnost pro začátečníky v práci s virtuální realitou
- Přirozené ovládání aplikace

2.2.1 Mischmasch

Mischmasch je jedním z nejlépe zdokumentovaných projektů. Vznikly kolem něj 2 studie [8] [9]. Mischmasch je velice slibný projekt ve fázi vývoje se zaměřením na multiplayer funkctionalitu. Autorům se podařilo vytvořit imerzivní prostředí, ve kterém z jednotlivých modulů staví několik lidí najednou hudební nástroj. Jejich inspirací je vize Jarona Laniera o kolektivní improvizované tvorbě virtuálních světů [8].

Tato práce nedosahuje estetické úrovně, o niž se snažím, a to z toho důvodu, že na grafickou podobu se zatím nezaměřuje. Mohu se z ní ovšem inspirovat implementací modulární stavby, kde všechny knoflíky modulů lze napojovat na další moduly (viz obrázek 2.4). Z volně dostupných videí je poznat, že i zvuková stránka je na dobré úrovni. Prostředím je neměnná černá plocha. Moduly mají velice jednoduché a srozumitelné rozvržení s několika málo parametry, vhodné pro začátečníky. A testujícím nevadilo trávit v simulaci delší časové úseky [9]. Práce s kabely je také zajímavá - samotné otáčecí knoflíky jsou vstupy a výstupy pro kabely. V dokumentaci projektu nejsou zmíněny výsledky testování z pohledu příjemnosti ovládání.



Obrázek 2.4. Ukázka z aplikace Mischmasch [9].

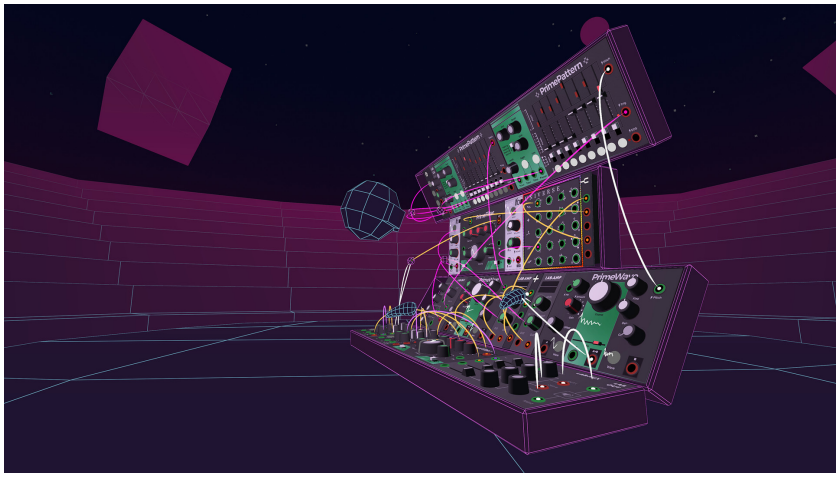
2.2.2 SynthVR

Dva nejpopulárnější projekty modulární syntézy ve virtuální realitě jsou SynthSpace a SynthVR [1]. SynthVR je projekt od autora Daniela Rothmanna zhotovený za karantény

[1] Vyhledávání na Google ke dni 16.12.2023 je zobrazí nejdříve.

2020 inspirovaný Eurorackem [10]. SynthVR má moduly rozvržením velice realistické, kterým ale chybí povrch odpovídající skutečným materiálům. Modularita i zvuk jsou úspěšně implementovány v enginu Unity [11].

Aplikace SynthVR autora Daniela Rothmanna má 86 procent kladných recenzí na platformě Steam[2]. Mezi slabiny projektu podle recenzentů patří málo grafické kustomizace a málo kustomizace herního prostředí. Dále porušuje jednu z korektních návrhových praktik a to jasnost stavu aplikace (nejsou zřejmé hodnoty na modulech). Také se v kritice objevuje absence takzvané freelocomotion. Jedna z návrhových praktik ve virtuální realitě pro snížení nevolnost je přenechání volby typu pohybu na uživateli. Ovládání je tedy přirozené až na chození samotného uživatele. Z recenzí si mohu odnést, že uživatelé si přejí větší volnost v upravování prostředí i modulů a případně propojení těchto dvou částí aplikace.



Obrázek 2.5. Ukázka z aplikace SynthVR převzatá ze Steamu [12].

2.2.3 SynthSpace

SynthSpace od Markuse Hofera je konkurent SynthVR a implementuje lepší podobu modulů a možnost práce v různých prostředích. SythSpace je vyvíjen v Unity [13] a dává uživatelům možnost vytvářet vlastní moduly pomocí Githubu [14]. Moduly jsou po stránce rozvržení i povrchu naprosto realistické a jejich podobou se budu inspirovat. Modularita i zvuk pracují výborně a navíc je implementována vizuální reprezentace zvuku demonstrující vliv modulů na signál. Mimo jiné obsahuje aplikace virtuální piano, možnost importování i exportování zvuků a sdílení projektů.

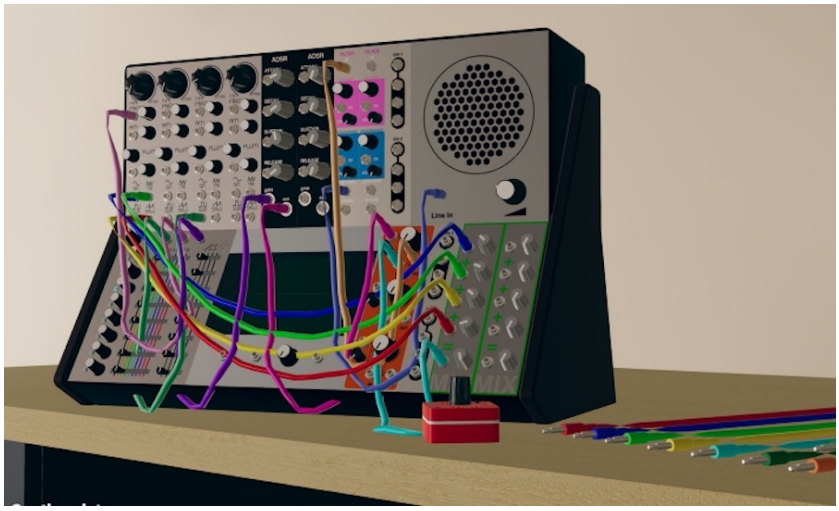
Z 34 recenzí na Steamu jsou 3 záporné [3]. Mezi stížnostmi je absence propojení s midi nástroji, absence možnosti nahrávání přímo do některého hudebního softwaru, absence tlačítka undo a nedostatečná nabídka efektů.

[2] Vyhledávání na `https://store.steampowered.com/app/1517890/SynthVR/` ke dni 16.12.2023.
[3] Vyhledávání na `https://store.steampowered.com/app/1355640/SYNTHSPACE/` ke dni 16.12.2023.

Obrázek 2.6. Ukázka z aplikace SynthSpace převzatá ze Steamu [15].

2.2.4 Synthmulator

Dalším menším projektem je Synthmulator vyvinutý v Unity s hotovým, Eurorack inspirovaným PROTO-1 syntezátorem od Oscara Sebia [16]. Synthmulator je zasazený ve statickém prostředí pokoje a moduly jsou napevno zapojené do sebe. Nenaplňuje tedy modularitu. Obsahuje však osciloskop a základní moduly pro modulární syntezátor. Po grafické stránce jsou knoflíky i vstupy pro kabely velice podobné Eurorackům, ale povrch nevypadá realisticky. Hráč se může volně pohybovat, ale nevidí modely svých rukou, což není optimální pro příjemný zážitek ze hry. Ovládání modulů zastaví pohyb hlavy, což není optimální pro začátečníky s virtuální realitou vzhledem k neintuitivnosti tohoto pohybu.



Obrázek 2.7. Ukázka z aplikace Synthmulator převzatá z [17].

2.2.5 OpenSoundLab

Open-sourcový OpenSoundLab v augmentované realitě s možností binaurálního HRTF zvuku od L. Zellera a H. Barfusse je postavený na SoundStage VR od L. Olsona [18]. I když modely nevypadají realisticky, mají kreativní řešení zobrazování stavu knoflíků. Modularita je plně implementována - moduly v podobě kvádrů se spojují kabely, které vedou z jejich boků. Po zvukové stránce jsou zajímavé obzvlášť binaurální efekty. Prostředí je realizováno Meta Passthrough módem. Program je vhodný pro začátečníky díky jednoduchosti modulů. Knoflíky se ovládají intuitivním otáčením ruky.

2.2.6 Závěr hodnocení existujících implementací

Další hudební aplikace obsahující modulární syntézu v různých rozsazích jsou Virtuoso VR, PatchWorld, Fields a AuSynthAR. Nepopsal jsem je do hloubky, jelikož modulární syntéza není jejich hlavním zaměřením.

Z průzkumu aktuálních implementací ve světě modulárních syntezátorů ve virtuální realitě je zřejmé, že i přes několik slibných projektů, existuje mnoho nevyzkoušených cest, kterými se může budoucí vývojář vydat.

2.3 Návrhové směrnice

Následující kapitola se bude zaměřovat na osvědčené návrhové praktiky, kterými se nechám vést v implementaci.

Návrhové praktiky pro uživatelské rozhraní ve virtuální realitě je možné rozdělit podle různých kritérií. Například podle na obecné praktiky a ty specifické pro virtuální realitu, praktiky specifické pro různé části VR zážitku podle průběhu a mnoho dalších. V této práci se budu volně držet rozdělení Jasona Jeralda v knížce The VR Book: základní praktiky, vnímání, nepříznivé účinky na zdraví, obsah aplikace, interakce a iterativní návrh. Dále budu doplňovat jednotlivé kapitoly o informace z dalších zdrojů.

2.3.1 Základy

Jak se ukazuje, první zážitky s návrhem rozhraní jsou pro uživatele velmi důležité [19]. Je proto nutné dodržovat při vývoji následující praktiky, aby se nový uživatel s virtuální realitou lépe sblížil. Ukázalo se, že pro uživatele je příjemné vidět společné artefakty se skutečným světem jako jsou ruce a nohy, aby se necítil ztracený. Pro začátek je vhodné vyvíjet kratší zážitky a přenechat kontrolu nad jejich délkou uživateli. To znamená mít mimo jiné dobře přístupnou možnost uložení a vypnutí aplikace. Pro dosažení skutečně imerzivního zážitku je důležité nenutit uživatele sundavat headset [5].

2.3.2 Zdraví

Jedna z kritických částí vývoje virtuální reality je redukce takzvané cybersickness. To je jev, který zažívá mnoho uživatelů při nebo i několik hodin po virtuálním zážitku. Může se projevovat podobně jako cestovní nevolnost a zhoršuje celý zážitek [20]. Jason Jerald dokonce tuto cestovní nevolnost popisuje jako největší risk pro virtuální realitu [5]. Naštěstí stejně jako několik dalších zdrojů [21] [22] nabízí možnosti, jak tento negativní efekt redukovat. Vypíšu praktiky důležité pro svůj vývoj.

Za prvé uživatelům doporučím lehký, bezdrátový, příjemný náhlavní displej bez viditelného blikání, s rychlou snímkovou frekvencí a odezvou. Tím je například Meta Quest 2/3, na kterém vyvíjím. Uživatel sám následně musí nastavit displej podle vzdálenosti svých zornic. Zorné pole simulace by mělo odpovídat zornému poli displeje, v případě Meta Quest 2 je to 98° [4]. Pro minimalizaci nevolnosti je lepší stabilní snímková frekvence než vysoká. V mé implementaci se promítá do optimalizace kódu.

Virtuální obsah by neměl být blízko k uživateli. Textová pole není vhodné umísťovat do uživatelova průhledového displeje (HUD), jako tomu může být v konvenčních 2D simulacích. Lepší je umísťovat informace do prostoru kolem uživatele. Tmavší scény snižují nepříjemný vjem blikání. Různé žárovky či kontrolky by neměly blikat rychleji než s frekvencí 1 Hz. Dobrá praktika je zobrazit uživateli virtuální zábranu ve chvíli, kdy se vzdálí z vyhraněného herního prostoru. Scéna by měla při zobrazení a zmizení

[4] Vyhledávání na `https://risa2000.github.io/hmdgdb/` ke dni 5.11.2023.

vyjet a zajet do černé. Náhlé změny mohou být nepříjemné. Vhodné je rozdělit obsah na interaktivní část a stabilní pozadí s horizontem a body, pomocí kterých se může uživatel orientovat.

Co se vztahuje k ovládání, je nevynucování rychlých pohybů hlavou. Pro můj návrh to znamená stavba syntezátoru na stolu před uživatelem a ne kolem dokola, jako tomu je například v OpenSoundLab [18]. Nastavování modulů může trvat delší úseky času a kvůli tomu je vhodné vést uživatele k sezení. Z mých zkušeností při používání virtuální reality je největší podněcovatel nevolnosti pohyb a rotace, kterou neovládá uživatel. Až na teleportaci tedy v mém návrhu nejsou žádné jiné simulací ovládané pohyby. Ovládání jednotlivých artefaktů modulu musí být svižné, aby uživatel nemusel udržovat příliš dlouho jednu pozici těla. Vzhledem k zvukovému zaměření aplikace je v neposlední řadě důležité chránit sluch uživatele automatickou kontrolou intenzity.

2.3.3 Obsah

Největší výhoda a zároveň výzva ve virtuální realitě je zobrazovaný obsah. Uživatel má nejvíce možností z celé historie médií, jak zažít virtuální obsah. To vytváří nové úkoly pro vývojáře, jelikož všechny možnosti náhlavních displejů je velice náročné využít. Následují existující osvědčené návrhové praktiky.

Podstatné je vymýšlení metafor se skutečným světem. V mém kontextu je to přirozené otáčení knoflíku nebo například vizuální podoba celého syntezátoru. Chceme angažovat uživatele do obsahu a tím vyvolat emoce a útěk od reality. Kontraintuitivně nám k tomu může napomoct uvěřitelnost našich metafor. Raději chceme uvěřitelnost než fotorealismus. S čímž souvisí, že musíme myslet na zážitek, který vytváříme, a ne na možnosti technologie, kterou používáme. Nemá smysl využít všechny možnosti náhlavního displeje, když jednoduchý, ale správný návrh vytvoří lepší zážitek. Jako příklad bych využil eye-tracking na displeji Meta Quest Pro. Technicky by bylo možné vybírat ovládaný artefakt pohledem, ale na zážitek uživatele by to pozitivní dopad dle mého názoru nemělo.

Obecně je dobré demonstrovat, proč uživatel ve virtuální realitě je a co může dělat. Dále je dobré ukázat jasný cíl a základní struktura světa musí být samovysvětlující. Toho můžeme dosáhnout například minimalizací rozptýlení - pokud uživatel uvidí stůl, syntezátor, moduly a pozadí, je jasné, co musí udělat. Pro maximální imerzivnost chceme minimalizovat přerušování zážitku. Hlavní zážitek musí být příjemný a obohacující, protože chyby v hlavním zážitku neopraví nadstavba a rozšíření. Pokud syntezátor nebude pro uživatele příjemný, další moduly simulaci nepozvednou.

Jedna z osvědčených praktik je využití barvy k zvýraznění důležitého obsahu. V případě rozšíření simulace na více než jeden stůl se syntezátorem je doporučováno rozdělení scény na zóny podle typu obsahu. Pak je možné využít zvědavost uživatele v zobrazení obsahu v periferním vidění. Tak jako u tradičního uživatelského rozhraní jsou i ve virtuální realitě důležité kvalitní popisy všeho obsahu, který není samovysvětlující. V případě využívání rovných panelů (přední stěna syntezátoru) je možné je zaoblit z důvodu čtení i ovládání.

Co se avatara týče, je dobré dovolit uživateli personalizaci. Minimálně ruce a možná i celý avatar by měly být v simulaci viditelné. 3D zvuk reagující na otáčení hlavy napomůže k imerzivnímu zážitku. Pomůže, pokud proporčně bude velikost rukou, stolu a syntezátoru uvěřitelná.

Uživateli musí být jasný stav simulace. To znamená zobrazovat hodnoty u knoflíku. Estetika napomůže k imerzivnímu zážitku a já jsem si ji už v minulé kapitole vybral jako jednu z důležitých částí svého projektu.

2.3.4 Interakce

Pohlcující vlastnosti virtuální reality ovlivňují způsob, jakým uživatel předpokládá, že bude interagovat. Na rozdíl od konvenční interakce s médii implementuje virtuální realita mnoho nových možností. Uživatel se může pohybovat různými způsoby jako je teleportace a chůze kolem obsahu. Předpokládá skutečnému světu věrné interakce s objekty. Oproti tradiční myši a klávesnici má úplně nové vstupní metody jako gesta a pohybové ovladače. Tyto nové metody přinášejí příležitosti, ale i omezení. Mezi návrhové praktiky, které jsou pro můj projekt důležité, patří následující:

Využívání interakčních omezení napomáhá k uvěřitelnosti zážitku (dveře se mohou otáčet kolem osy - uživatel je nemůže vyndat a odložit pryč). Gravitace a podobná skutečná omezení nemusí nutně vylepšit zážitek. Není vhodné implementovat gravitaci pro kabely, které slouží k propojení modulů v modulárním syntezátoru, protože by to způsobilo delší dobu zotavení po chybě při manipulaci s kabely. Všechny zdroje, jako jsou návody a v mém návrhu další moduly, musí být dostupné uvnitř virtuální reality. Možnosti jsou vztah mezi uživatelem a obsahem ne vlastnosti obsahu. Jinými slovy, návrh se musí zaměřit na uživatele a na to, jaké interakce jsou pro něj vhodné. Pro ilustraci, oscilátory skutečných syntezátorů se musí ladit, a to většinou zevnitř. Přestože by taková interakce mohla být realistická, nebyla by pro uživatele praktická.

Rozhraní interakce musí být snadno pochopitelné. K tomu nám pomůže užití metafor se skutečným světem. Uživatel si tak vytvoří mentální model interakce. Chceme implementovat návrh zaměřený na člověka - okamžitá odezva, konzistentní možnosti, omezení vedoucí k správným akcím. Konzistentní interakční rozhraní vede k využívání naučených praktik v dalších částech simulace. Ačkoliv uživatel potřebuje zpětnou vazbu pro ujištění sama sebe, že to, co dělá, dělá správně, je nežádoucí zpětnou vazbou přehlcovat. I když je přímá interakce pravděpodobně první, kterou uživatel vyzkouší, měly by mu být nabídnuty další možnosti. Například prodloužení dosahu uživatele pomocí nástrojů. Přirozeně budou někteří uživatelé používat obě ruce najednou. Simulace na to musí být připravena. Ve shrnutí, realistické interakce pomáhají imerzivnosti a nerealistické k snížení únavy. Rozhodnutí, kterou používat, ale přenecháváme na uživateli.

Virtuální nástroje pokládáme blízko k uživateli - na dosah ruky. Uživatel by měl ve virtuálním světě vidět místo na případné odložení ovladačů (stůl se syntezátorem). Informace o stavu simulace by měl uživatel mít dostupné, ale neměli bychom mu je vnucovat k hlavě, ideálně položit kolem trupu. V případě implementace gest je nutné omezit jejich množství pro lepší zapamatování. Pokud je možnost více druhů interakce, měli bychom dát uživateli zpětnou vazbu o tom, kterou používá.

Přechod mezi výběrem objektu a interakcí by měl být hladký. Při výběru objektu je vhodné vizuálně zobrazit, který objekt by se právě vybral. Při vybírání malých objektů implementujeme přeskakování k nejbližšímu. V případě přímého posouvání objektu ve virtuálním světě je vhodné nechat při střetu s dalším předmětem objekt vnořit při hlubokém střetu, ale fyzicky omezit při střetu mělkém.

Pro zpříjemnění zážitku od zapnutí náhlavního displeje po jeho sundání se chceme vyvarovat nucení uživatele opravovat fyzickou pozici po otevření aplikace a nucení uživatele se fyzicky hýbat mezi úkoly. Pro pohyb chceme využít skutečnou chůzi, pokud je virtuální prostředí stejně velké jako prostředí uživatele. Uživatel musí mít na výběr ale také teleportaci. V případě rozšíření simulace o další pracovní stanoviště je vhodné implementovat waypointy pro urychlení práce. Haptické možnosti ovladačů jsou jedna z nejlepších zpětných vazeb, kterou můžeme uživateli poskytnout. Můžeme využít známé interakce z 2D medií. Vhodné je to například v ovládání menu. Ovládací panel s menu

chceme přilepit k nedominantní ruce. Na závěr, samozřejmá je možnost 360 stupňového otáčení a pohledu.

2.3.5 Iterativní návrh

Vzhledem k relativně krátkému uplatňování virtuální reality nejsou principy ve vývoji ještě uceleny. Proto je podstatné provádět iterativní návrh, kde se k návrhu vývojář vrací, hodnotí ho a podle poznatků upravuje další verze. V praxi se využívá zpětná vazba z testování s a bez uživatelů.

Před vytvářením návrhu chceme vyzkoušet mnoho různých aplikací virtuální reality. To nám pomůže zjistit, co je možné. Naplňované předpoklady musí být ověřitelné a stručné, ale s prostorem pro inovaci. Při tvorbě návrhu vstupujeme do role uživatele. Práme se co by pro něj bylo ideální. Musíme předejít analystické paralýze. Například tím, že ze začátku projektu budeme často selhávat. Pokud se od začátku neselhává, je žádoucí větší inovace.

Špatný ale funkční prototyp je lepší než debata. Při tvorbě návrhu chceme zapisovat důvod pro změny, kterými procházíme. Po vydání je vhodné zpětně se vracet k představě cílové skupiny podle skupiny, která aplikaci skutečně využívá.

Stejně důležité, jako popsat, co aplikace bude obsahovat, je důležité popsat to, co obsahovat nebude. Během procesu navrhování se musíme soustředit na výhody a výsledky, spíše než na vlastnosti návrhu a před rozhodnutím prozkoumat různé návrhové možnosti. Pokud to prostředí povoluje, chceme využít existující frameworky a nástroje. Po vytvoření návrhu definujeme případy užití pro určení interakcí potřebných v implementaci. Implementovat a optimalizovat jednu možnost před přidáním alternativ.

Ve chvíli, kdy se dostaneme do fáze testování, chceme získat zpětnou vazbu co možná nejrychleji. To znamená, že testovat s minimálními prototypy chceme co možná nejdříve ve fázi vývoje. Zpětnou vazbu chceme získat od co nejvíce naivních uživatelů (mají minimální zkušenosti s naším prostředím).

Osobně je nutné si udržovat pozitivní vztah ke konstruktivní kritice, při vyhodnocování testování neobviňovat uživatele a nezaseknout se v získávání jednoho druhu zpětné vazby. Chceme se ptát různých uživatelů na různé části zážitku. Nejdříve získáváme kvalitativní data, pak až kvantitativní. Měření se provádí v různých časech, místech a s různými skupinami. Před každým sezením se musíme ujistit, že měření měří to, co chceme měřit. Chceme znemožnit testerovi uhádnout naši hypotézu, aby jí nebyl ovlivněn. Interview fungují nejlépe hned po zážitku v simulaci. U každého testera musíme ověřit, že skutečně patří do námi vymyšlené cílové skupiny.

2.3.6 Shrnutí návrhových praktik

Pro můj projekt vychází z návrhové analýzy několik jednoznačných doporučení. Tato doporučení mi napomohla k vytyčení cesty v dalším vývoji.

Při ovládání modulů se musím co nejvíce přiblížit ovládání skutečného modulárního syntezátoru. Pro pohyb po místnosti se syntezátorem musí být různé možnosti. Aplikace v enginu musí být optimalizována pro stabilní snímkovou frekvenci. A mnoho dalších exaktních zužitkování představených návrhových praktik.

Kapitola 3
Návrh

Následující kapitola představí koncepty, ať už vizuální nebo funkční, které jsem před implementací stanovil v závislosti na cílové skupině a zadání této práce.

3.1 Cílová skupina

Hlavní cílová skupina obsahuje umělce (sound designer, performer, muzikant), kterého zajímají hudební možnosti modulárních syntezátorů, ale tento nástroj je pro něj nedostupný. Skutečná jednotka je příliš drahá na příležitostní zkoušení. Je často těžká a příliš rozměrná. Umělec chce před investicí vědět, zda pro něj bude modulární syntéza přínosná. Další alternativa je pak desktopová simulace, ale ta nedosahuje požadované autenticity. Pocit ze skutečného, velkého přístroje, vkládání libovolných modulů v libovolném pořadí do racku a otáčení velkých knoflíku pro dosažení požadovaného výsledku jsou všechno důležité součásti modulární syntézy.

Virtuální realita je tedy pro umělce perfektním kompromisem. Kombinuje interakce odpovídající fyzické jednotce se všemi výhodami virtuálního headsetu (dostupnost, nezávislost na místě, přenosnost a další). Cílový uživatel se nesnaží nahradit svůj aktuální modulární systém.

3.2 Příklad uživatele

Vysokoškolský student Ondřej. Je volnočasový muzikant a hráč videoher. Má základní znalosti o modulárních syntezátorech a o ovládání virtuální reality. Má nápady na nové syntezátory a chce je v příjemném prostředí zkusit sestavit. Následně by si přál vyzkoušet vystoupení s takovým zařízením. Předpokládá intuitivní ovládání, možnosti ukládání své práce a přirozenou vizuální podobu aplikace.

3.3 Storyboard obsahující příkladného uživatele

Ondřej dostane nápad na novou konfiguraci modulů v modulárním syntezátoru a s představou zvuku, kterého chce dosáhnout, zapíná virtuální realitu. Otevře nebo založí projekt. Nastaví základní moduly jako oscilátor, LFO, obálky a výstup do těla nástroje. Interaguje v průběhu testování s moduly a v reálném čase upravuje zvuk. Ukládá projekt a vypíná virtuální realitu.

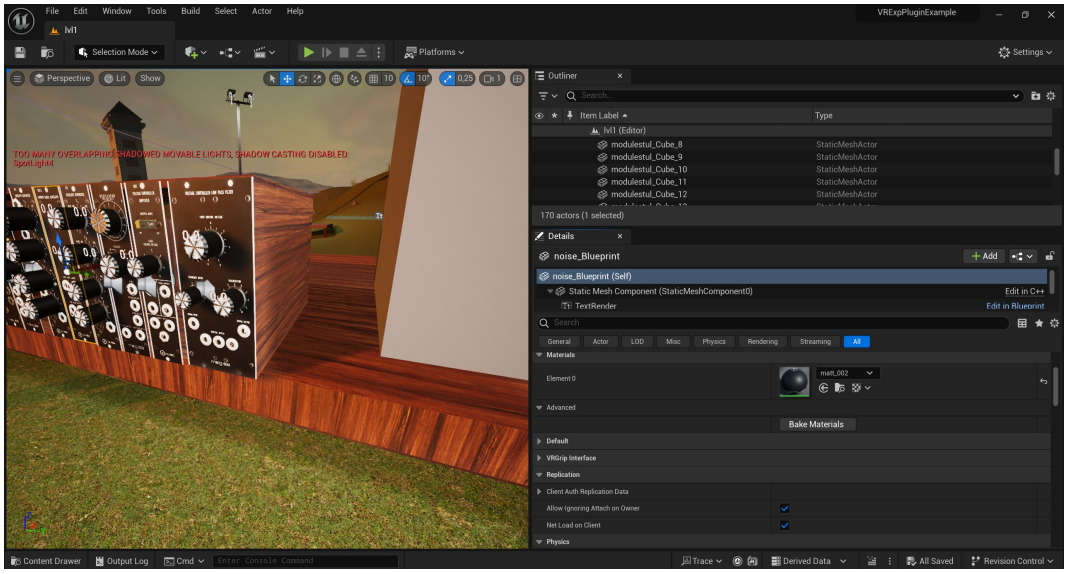
3.4 Návrh platformy

První krok návrhu byl výběr herního enginu. Jelikož jsem před touto prací měl zkušenosti pouze s Godotem, který se zdál pro virtuální realitu nevhodný, vybíral jsem mezi Unity a Unreal Enginem. Ačkoliv Unity má daleko lepší dokumentaci různých projektů

virtuální reality, kvůli myšlence jednoduché efektní grafiky, jsem se rozhodl pro Unreal Engine. V době návrhu navíc Unity ztratilo svými rozhodnutími ohledně monetizace popularitu a Unreal Engine vzbuzoval pocit silnějšího uplatnění v budoucnosti.

Unreal Engine byl vytvořený studiem Epic games pro FPS žánr a rychle se stal kompetitorem her založených na Quake. Je napsán v jazyce C++ a s nejnovější verzí 5.3 je obecně přijímán jako jeden z nejlepších na trhu.

V roce 2023 si držel přibližně 30 % videoherního trhu [23]. Vznikly v něm populární tituly jako Fortnite, Borderlands 3, Batman Arkham Knight nebo Gears of War. Vývoj v Unreal Enginu je pro nekomerční užití zdarma a i díky tomu je to výborný nástroj na prototypování.



Obrázek 3.1. Ukázka uživatelského prostředí Unreal Enginu.

Hry se v něm programují v jazyce C++ a pomocí vizuálních Blueprintů. Ty napomáhají k ještě rychlejšímu prototypování a otevírají nové možnosti pro návrháře nezkušené v C++. Má vlídné uživatelské prostředí a v komunitě je chválen za schopnost vytvářet velice kvalitní vizuály [2].

3.5 Návrh pro funkci aplikace

Produkt, který vytvářím, musí splňovat následující funkční požadavky.

1. 10 audio patchů vytvořených v jazyce Pure Data
2. Věrné napodobení modulárního systému a kvality zvuku z něj vycházející
3. Přístupnost v brýlích s virtuální realitou
4. Jednotlivé moduly nástroje zapouzdřují Pure Data patche
5. Spojování modulů pomocí kabelů
6. Generace kabelů a modulů pomocí menu
7. Ukládání

Pure Data patche jsou zapouzdřením v modulech pro uživatele nedosažitelné. Výsledkem je dojem, že zvuk skutečně tvoří rozhraní, se kterým uživatel interaguje.

3.6 Návrh pro interakci a zpětnou vazbu aplikace

Nástroj musí mít intuitivní interakce. Všechny procesy viditelné pro uživatele jsou analogické skutečné práci s modulárním syntezátorem. Z toho důvodu, na rozdíl od modulů v Mischmasch, musí mé moduly odpovídat skutečným jednotkám. Propojování modulů se implementuje pomocí kabelů zapojených do vstupů na přední straně modulů. Knoflíky modulů se otáčí stiskem zadního analogového tlačítka na ovladači (haptická zpětná vazba) a otočením ruky. Podobným pohybem je možné moduly chytit do ruky a umístit jinam. Tyto pohyby musí být začátečníkovi představeny pomocí statického textu vedle syntezátoru. Uživatel slyší zvuk pouze v případě, že je reproduktor připojen na správný modul.

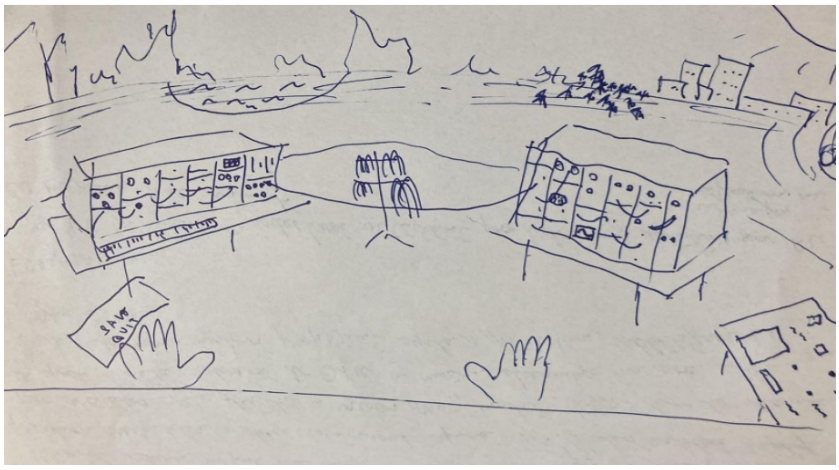
Pro zajištění příjemné práce v simulaci se na několika místech musí implementovat zacvaknutí. Kabel se zacvakne do vstupu modulu. Moduly se zacvaknou k přední straně těla syntezátoru. Nakonec se i knoflíky s diskrétními hodnotami po puštění zastavují na jednotlivých hodnotách.

Vzhledem k povaze této aplikace je otáčení knoflíkem snad nejdůležitější interakce, kterou uživatel prožívá. Knoflík dává zpětnou vazbu v podobě virtuálního textu s aktuální hodnotou rotace (popř. s jiným informativním textem). Text se objevuje v moment uchycení (v reakci na negativní kritiku SynthVR).

Kompromis autenticity, který ale ulehčí zážitek, je opomíjení gravitačního zrychlení pro moduly a kabely.

3.7 Návrh pro UI

Uživatel z cílové skupiny chce zpravidla simulovat pocit z práce se skutečným modulárním syntezátorem. Je tedy velký nárok na realismus textur. I rozvržení modulů by mělo odpovídat původním modulům od firmy Moog 2.3. O tom více v kapitole 3.9. Vizuální stránka modulů odpovídá skutečným. Do těla se připojují způsobem, kde se obdélník na výšku postaví do strany těla. Kabely budou mít rozdílné barvy, díky kterým bude patchování (propojování modulů) příjemnější (v reakci na negativní kritiku SynthVR). Prostředí simulace by mělo být příjemné. Uživatel má možnost toto prostředí prozkoumat několika druhy pohybu. Správně umístěné zdroje světla napomohou uvěřitelné atmosféře a zároveň mohou uživatele upozornit na důležité části světa. Interakční objekty se snaží o realismus, informační prvky, jako je text nebo menu, musí být ale jasné a dobře čitelné. Jednotlivá tlačítka menu budou orientována hodinovým způsobem kolem ruky uživatele. Z analýzy víme, že uživateli se lépe orientuje, když má viditelné ruce. Další prvky simulace napomáhající orientaci budou neměnný obzor, statické prvky světa (budova v dáli, jezero v blízkosti a hory) a zvuk vycházející z reproduktoru, který bude působit směrově.



Obrázek 3.2. Sketch prostředí z prvních stádií návrhu.

V prostředí by měla být stanice k práci (box syntezátoru), u které se uživatel objevuje. Uživatel je tedy v příjemném, přirozeném prostředí a prvky, které vidí, mu připadají skutečné.

3.8 Inspirace pro moduly

Ačkoliv momentálně populární řešení modulárních syntezátorů jsou systémy Eurorack (pro možnosti výběru mnoha výrobců a dostupných cen), rozhodl jsem se založit své moduly na původních ikonických výrobcích firmy Moog. Jsou jednotlivě velice dobře zdokumentovány [1]. Hlavním důvodem je fakt, že pro mnoho lidí představují podstatu modulárních syntezátorů. Vedlejším důvodem je ucelenost jejich návrhu. Moderní moduly do systému Eurorack často inovují a přináší nové a nové funkcionality. Ale právě původní systémy od Moogu dávají základ hlavnímu rozvržení modulárního syntezátoru a základním modulům, které v něm nesmí chybět.

Obrázek 3.3. Moog System 55 obsahující podobné moduly jako Model 10 .

Pro mých 10 modulů je tedy výchozím bodem po straně funkcionalit, vizáže i interakce Moog Model 10 [2]

[1] `https://modularsynthesis.com/` má články o všech originálních modulech 60. a 70. let.

[2] Nová jednotka tohoto jména Vás ke dni 17.04.2024 vyjde na necelých 364 000 Kč `https://www.moogmusic.com/products/model-10`.

3.9 Moduly

V této pasáži vyjmenuji 10 návrhů modulů a jejich opodstatnění. Popis půjde přibližně cestou, kterou signál v konvenčních nastaveních následuje. Vzhledem k médiu, na kterém bude aplikace spouštěna, není žádoucí, aby moduly obsahovaly mnoho interakčních prvků poblíž sebe. Z toho důvodu je cílem každého modulu jednoduchost a korektní plnění jedné jednoznačné funkce. Uživatel se díky tomuto konceptu nemusí trápit s neúmyslným uchycením malých knoflíků a složitější funkce může složit využitím neomezeného množství modulů.

3.9.1 VCO

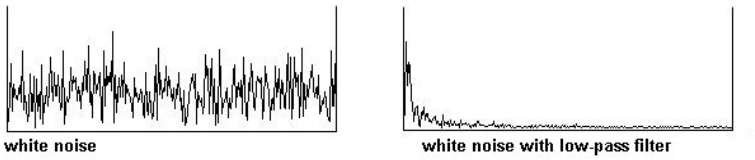
Jako první je zpravidla generátor signálu. V mém případě je to oscilátor (VCO - Voltage Controlled Oscillator). Oscilátor má jeden vstup a výstup. Je to modul založený na jednotce 921 B a má tedy dva knoflíky. Jeden přidá až +12 půltónů respektive -12 půltónu s rovnoměrným přesunem mezi jednotlivými hodnotami. Druhý představuje rozsah dále upravovaný prvním knoflíkem. Je definován po vzoru varhanních spínačů a má 6 možných úrovní. 5 z nich má pojmenování po původních výškách píšťal a poslední (respektive první z levé strany) představuje nastavení LFO - Low Frequency Oscillator, které se zpravidla používá na pomalou modulaci hodnot jiných modulů. Další hodnoty jsou 32', 16', 8', 4' a 2'. Frekvence jim odpovídající můžete vidět v tabulce 3.1.

Výška	Frekvence
LOW	5.2 Hz
32'	65.4 Hz (C2)
16'	130.8 Hz (C3)
8'	261.6 Hz (C4)
4'	523.2 Hz (C5)
2'	1046.5 Hz (C6)

Tabulka 3.1. Seznam frekvencí odpovídajících jednotlivým výškám píšťal. Hodnoty převzaty z [24].

3.9.2 White noise

Bílý šum je jednoduchý modul s jedním výstupem a žádným ovládáním. Je založený na jednotce 903 A a generuje zvuk s rovnoměrnou hlasitostí všech slyšitelných frekvencí (20 - 20 000 Hz). Na obrázku 3.4 je vidět frekvenční spektrum takového signálu.

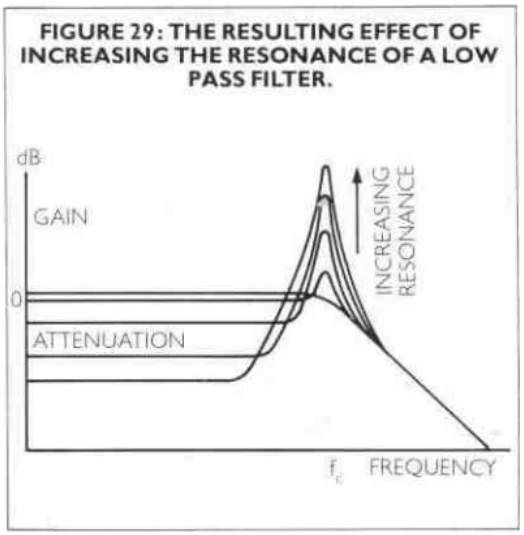


Obrázek 3.4. Ukázka frekvenčního spektra bílého šumu [4].

3.9.3 VCF

Dále zmíním VCF (Voltage Controlled Filter). Jedná se o filtr s dolní propustí založený na jednotce 904 A. Má dva vstupy a jeden výstup. Jeden vstup je pro audio signál, který chceme filtrovat, a druhý pro zdroj modulace frekvence, pod kterou filtr propouští.

Tato frekvence je určována zároveň velkým knoflíkem s rozsahem 20 - 20 000 Hz. Druhý knoflík ovládá resonanci. Ta dále upravuje tvar obálky filtru. Tvoří výšku vrcholu před utnutím. Má rozsah 0 - 10 a při hodnotách > 8 dochází k samo-oscilaci (feedback smyčka implementující resonanci posílá až příliš signálu zpátky na začátek a signál nikdy nevyprchá).



Obrázek 3.5. Ukázka demonstrující obálku tvořenou low pass filterem s resonancí [7].

Do návrhu není zahrnut třetí knoflík z původního modulu 904 A, aby mohl uživatel jedním pohybem dosáhnout na celý rozsah 20 000 Hz.

3.9.4 VCA

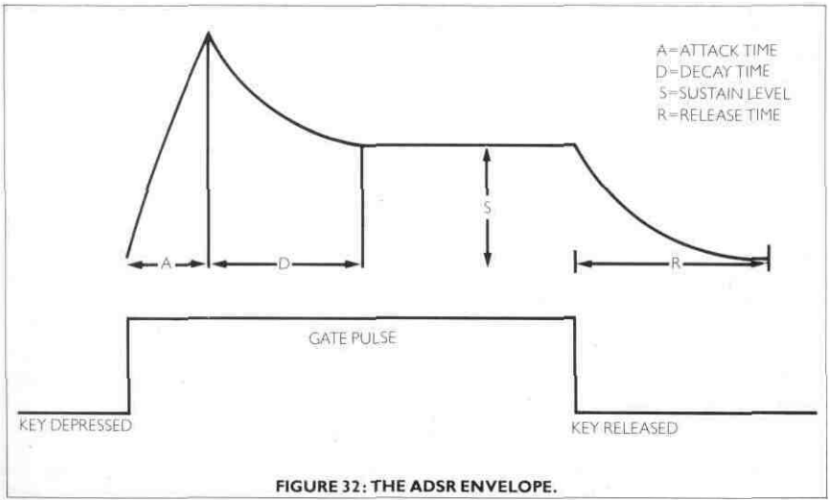
Podobný modul je VCA (Voltage Controlled Amplifier). Má identické vstupy a výstupy jako předchozí modul a je navržen podle jednotky 902. Jeho jediný knoflík násobí signál a tím snižuje a zvyšuje jeho amplitudu. Tato amplituda může být modulována signálem z druhého vstupu.

3.9.5 Mixer

Důležitý modul je mixer. Původní jednotky plnící tuto funkci jsou pro důvody zmíněné na začátku této kapitoly moc složité. Modul realizuje kombinaci čtyř vstupních signálů do jednoho výstupního. Každý jednotlivý vstup má přiřazený knoflík upravující jeho amplitudu. Na rozdíl od VCA nemá tento modul vstup pro modulaci.

3.9.6 ADSR

Modulační vstupy se často plní LFO a pak takzvanou ADSR obálkou. A - attack, D - delay, S - sustain, R - release. ADSR modul má jeden výstup po vzoru jednotky 911. Dále má 4 knoflíky odpovídající odshora jednotlivým písmenům. Po impulzu vytvoří křivku tvarovanou pomocí hodnot na knoflících. Vzroste na maximální hodnotu v čase A, klesne na úroveň definovanou S za čas D a následně klesne na hodnotu 0 za čas R, dle schématu 3.6.



Obrázek 3.6. Ukázka křivky definované ADSR modulem [7].

Tato křivka po zapojení např. do VCA dokáže simulovat změny hlasitosti zahrané struny kytary v čase.

3.9.7 Sequencer

Zatímco výška tónu u tradičních hudebních nástrojů může být udávána různými klávesami piana nebo pražci na krku kytary, tento systém zatím určuje výšku pomocí prvního knoflíku na VCO nebo modulací na vstupu VCO. Pro navýšení hudebního potenciálu se z toho důvodu do modulárních systémů často přidává tzv. sequencer. Definuje sekvenci několika (často pevně definovaný počet) tónů za sebou. Každý tón má výšku a hlasitost. Tento modul je volně inspirován velmi složitou jednotkou 960. Má knoflík ovládající rychlost přechodu jednoho tónu na následující, knoflík ovládající individuální hlasitost pro každý z osmi tónů v sekvenci a knoflík ovládající jeho výšku. Dále dva výstupy pro výšku a hlasitost zvlášť.

3.9.8 FX

Syntezátory (nejen modulární) bývají často využívány v kombinaci s různými efekty (reverb, delay, phaser, chorus, distortion). V reakci na negativní kritiku SynthSpace jsem se tedy pro zajištění kompletního zážitku rozhodl navrhnout dva základní efekty, a to delay a reverb. Tyto efekty mají modul velice podobný oscilátoru, ale v původních Moog systémech zahrnuty nebyly. Delay vstupní signál několikrát opozdí a tyto opožďované zvuky spojuje se signálem původním. V přírodě se s ním můžeme setkat například při ozvěně v horách. Reverb zvuk opozdí podobně jako delay, ale na rozdíl od efektu delay dochází velikému množství těchto opoždění a není možné mezi nimi rozlišovat. Setkat se s ním můžeme například ve velkých katedrálách. Oba mají jeden vstup pro signál a výstup pro signál kombinovaný s efektem. Mají také 2 knoflíky ovládající vlastnosti svého efektu. Delay obsahuje knoflík definující délku trvání a druhý knoflík definující barvu vracejícího zvuku. Reverb má analogické rozložení.

3.9.9 Theremin

Poslední modul není tak docela modul. Je to rozhraní schopné definovat výšku a hlasitost stejným způsobem, který implementuje skutečný theremin. Úrovně jsou separátně udělovány vzdáleností rukou od dvou tyčí v horizontální a vertikální orientaci.

Ačkoliv se může zdát theremin jako nemístný nápad to projektu zabývajícího se modulární syntézou, věřím, že je tomu přesně naopak. Právě theremin vynalezený Leonem

Thereminem, byl prvním hudebním nástrojem, který Bob Moog uvedl na trh v podobě zobrazené na obrázku 3.7.

Obrázek 3.7. Ukázka prvního nástroje, který Bob Moog prodával [25].

Thereminy mají tedy podobné kořeny s modulárními syntezátory. Navíc na rozdíl od jiných hudebních nástrojů, které bychom mohli ve virtuální realitě tvořit, zůstává způsob ovládání virtuálního a skutečného thereminu velice podobný.

3.9.10 Speaker

V poslední řadě se na konci cesty signálu musí objevit modul reprodukující vytvořený signál. Vlastní pouze jeden vstup a přijímaný signál jednoduše reprodukuje do světa.

Kapitola 4
Implementace

Kapitola implementace postupně představí fáze vývoje od propojení Pure Dat a Unreal Enginu přes různé kódy podporující simulaci až po tvorbu objektů scény simulace.

4.0.1 Zprovoznění platformy

Instalace Unreal Enginu pomocí platformy Epic byla přímočará a engine nevyžadoval další nastavování. Obsahuje dokonce připravený level pro virtuální realitu.

V dalším kroku jsem se seznamoval s brýlemi Oculus Quest 2 zapůjčenými fakultou. The VR Book doporučovala před implementací vlastního projektu vyzkoušení různých aplikací, které daná virtuální realita nabízí, a i tím jsem se tedy zabýval [5]. Integrace s Unreal Enginem byla až na výjimky jednoduchá a rychle jsem došel k závěru, že kabelové propojení, jakkoliv omezující, nabízí stabilnější připojení, než to bezdrátové.

Interakci brýlí a enginu ulehčil plugin VR Expansion Plugin (VRE) [26]. Obsahuje funkce, které může vývojář volat, přímo napojené na ovladače a vývoj je díky němu přístupnější. Obsahuje také ukázkový level s různými druhy interakcí implementovaných pluginem. Právě v něm jsem experimentoval s implementací návrhu představených komponentů.

4.1 Propojení Pure Dat a Unreal Enginu

Výrazně náročnější než všechny ostatní části této práce bylo propojení Pure Dat a Unreal Enginu. Narazil jsem na mnoho předem nepředvídatelných překážek, které jsem plně překonal až po měsíci a půl práce. V této sekci se je pokusím prezentovat.

Před začátkem implementace jsme zvažovali několik cest, kterými se vývojář s tímto problémem může vydat.

Za prvé může zkompilovat Pure Data patch pomocí kompilátoru Heavy a následně z něj udělat VST plugin pomocí WWise, který podporuje spolupráci s Unreal Enginem [27].

Dále je možné ovládat instanci Pure Dat pomocí síťové komunikace.

Nakonec cestou, která se prokázala pro nás za nejlepší, bylo obalení patchů pomocí knihovny LibPD a její instanciování v C++ kódu jednotlivých tříd Unreal Enginu.

4.1.1 Heavy a WWise

Tato cesta má jeden kvalitně zpracovaný návod [27]. Naneštěstí jsem při svém pokusu narazil na mnoho problémů způsobených verzováním. Kompilátor Heavy má online verzi, která má bohužel problémy s častými výpadky. Existuje i offline verze [28]. Po jejím zprovoznění jsem zkompiloval Pure Data patch s přepínačem -g wwise. Tento zkompilovaný kód bohužel WWise neviděl kvůli nekompatibilním verzím. Při pokusu propojení WWise a hvcc ve starších verzích obou programů se ukázalo, že výsledný plugin by s aktuální verzí Unreal Enginu (5.3) nespolupracoval.

4.1.2 Komunikace uvnitř Pure Dat

Další možnou cestou je využití objektů netsend v Pure Datech. Pomocí UDP nebo TCP protokolu a IP adresy dokáže tak Pure Data posílat zprávy. Vzhledem k různým modulům, které chceme v práci vytvořit, je vhodné mít několik jednotlivých instancí Pure Dat vznikajících v průběhu. To by tento způsob implementace neumožnil.

4.1.3 LibPD

LibPD je knihovna napsaná v jazyce C vytvořená původně v roce 2010 za účelem spouštění Pure Data patchů na Androidových zařízeních. Umožňuje zabalení jazyka Pure Data (v základní verzi pouze vanilla objekty) pro různé další jazyky a stále se vyvíjí [29]. Z pohledu implementace ale i LibPD cesta přinesla několik komplikací. Pro nezkušeného uživatele byl build knihovny s podporou více instancí náročný. Opravování chyb bylo také náročné, protože i špatně postavená knihovna dovolovala zdánlivě bezproblémový běh programu. Nepomáhal fakt, že pro build měl uživatel několik možností (Cmake a Visual studio, Makefile a další). Po neúspěšných pokusech o build Makefilu pomocí různých způsobů (Choco instalace Make) pomohl komplikace spojené s operačním systémem Windows vyřešit MinGW systém. Následuje návod na build v takové podobě, ve které se LibPD využívá v této práci.

1. Instalace MinGW
2. Vytvoření a navigace do složky v msys64/home/user/
3. Otevření msys terminálu v této složce

Pokračuje vložení následujících příkazů do tohoto terminálu:

```
> git clone --recurse-submodules https://github.com/libpd/libpd.git
> cd libpd
> git checkout 0.12.1 (nebo jiná poslední verze z tagu na githubu)
> vcpkg new --application
> vcpkg add port pthreads
> vcpkg install
```

V souboru CMakeLists.txt změníme option(PD_MULTI Compile with multiple instance support OFF) na option(PD_MULTI Compile with multiple instance support ON). Nakonec znovu vložíme příkazy do spuštěného terminálu:

```
> mkdir build
> cd build
> cmake ..
> -DCMAKE_THREAD_LIBS_INIT:PATH=
"your_path_to_this_folder\libpd\vcpkg_installed\x64-windows
\lib\pthreadVC3.lib" -DPTHREADS_INCLUDE_DIR:PATH=
"your_path_to_this_folder\libpd\vcpkg_installed\x64-windows\include\"
> cmake --build .
```

4.1.4 Vložení LibPD do Unreal Engine projektu

Po kompilaci knihovny získáme dva soubory - libpd-multi.dll a libpd-multi.lib (multi značí podporu více instancí LibPD, což je pro náš projekt velice důležité). Dll v prostředí Windows znamená dynamickou knihovnu. Lib slouží při kompilaci k definici funkcí, které kompilátor do spustitelného souboru připojí. Tyto soubory, ale není možné jednoduše vložit mezi C++ třídy modulů kvůli způsobu, kterým se Unreal Engine kompiluje.

Unreal Engine používá UnrealBuildTool, který pomocí souborů napsaných v jazyce c# (build.cs) kompiluje různé části projektu (např. pluginy).

Naším cílem je vytvořit takový plugin ze zkompilované knihovny LibPD. Využíváme k tomu příklad práce s knihovnou třetí strany, dodaný Unreal Enginem. Tento example plugin z enginu vytvoříme a následně do něj zabudujeme LibPD. LibPD bude v pluginu ve složce plugin/Source/ThirdParty/. Za prvé musí Build.cs tohoto pluginu připsat LibPDLibrary do svých závislostí.

```
PublicDependencyModuleNames.AddRange(new string[]{"Core","CoreUObject",
"Engine","AudioMixer","LibPDLibrary","Projects"});
```

Samotné soubory .lib a .dll budou umístěny do složky plugin/Source/ThirdParty/LibPDLibrary/x64/Release. Do této složky musíme také v našem případě připojit knihovnu pthreadVC3.dll, ze které LibPD čerpá. V Build.cs knihovny (tedy ne v pluginovém Build.cs) musíme UnrealBuildToolu říct, jak má s knihovnou nakládat a kde ji najít.

```
Type = ModuleType.External;
PublicSystemIncludePaths.Add("$(ModuleDir)/Public");
if (Target.Platform == UnrealTargetPlatform.Win64)
{
// Přidání .lib knihovny se jmény funkci
PublicAdditionalLibraries.Add(Path.Combine(ModuleDirectory,
"x64","Release","libpd.lib"));

// Informace, že nechceme .dll knihovnu načítat při spouštění aplikace,
// ale naopak až ve chvíli, kdy bude nezbytná
PublicDelayLoadDLLs.Add("libpd.dll");

// V případě, že se bude projekt exportovat do spustitelného souboru,
// chceme do něj zahrnout i celé LibPD a pthread
RuntimeDependencies.Add(
"$(PluginDir)/Binaries/ThirdParty/LibPDLibrary/Win64/libpd.dll");
RuntimeDependencies.Add(
"$(PluginDir)/Binaries/ThirdParty/finalpdLibrary/Win64/pthreadVC3.dll");
}
```

V posledním kroku otevíráme plugin.cpp soubor ve vnějším pluginu (ne tedy v LibPDLibrary).

```
void FfinalpdModule::StartupModule()
{
// Tato metoda je volána Unreal Enginem v moment, kdy načítá náš plugin

// Zde definujeme cestu ke knihovnám (tedy i pthread)
FString BaseDir = IPluginManager::Get().FindPlugin(
"finalpd")->GetBaseDir();

FString LibraryPath;

LibraryPath = FPaths::Combine(*BaseDir,
TEXT("Binaries/ThirdParty/LibPDLibrary/Win64/libpd.dll"));
```

```
FString LibraryPath2;
LibraryPath2 = FPaths::Combine(*BaseDir,
TEXT("Binaries/ThirdParty/finalpdLibrary/Win64/"));
FPlatformProcess::AddDllDirectory(*LibraryPath2);

// Načítáme LibPD (už bez zpoždění) a získáváme handler knihovny
ExampleLibraryHandle = !LibraryPath.IsEmpty() ?
FPlatformProcess::GetDllHandle(*LibraryPath) : nullptr;

if (ExampleLibraryHandle)
{
}
else
{
FMessageDialog::Open(EAppMsgType::Ok, LOCTEXT("ThirdPartyLibraryError",
"Failed to load example third party library"));
}
}
```

Po těchto změnách vložíme zkompilovanou knihovnu společně s pthread knihovnou do složky plugin/Binaries/ThirdParty/LibPDLibrary/Win64 a vše je připraveno.

4.1.5 Konečná podoba propojení

Zkompilovaná knihovna LibPD existuje jako plugin v Unreal projektu. A v třídách Unreal Enginu je možné volat metody LibPD na různých instancích.

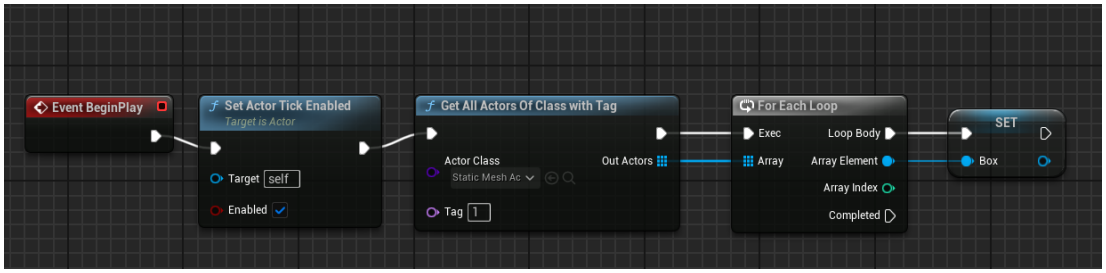
4.2 Kód projektu

Následující kapitola představí esenciální části kódu společně s vysvětlením, jaké funkce zprostředkovávají.

4.2.1 Blueprinty

Jak už bylo zmíněno, Unreal Engine dodal funkcionalitu grafického programování v Blueprintech, která umožňuje high-level přístup k programu. Každá třída má svůj vlastní Blueprint, který obsahuje klasické části objektově orientovaného programování jako konstruktor, vlastní proměnné, funkce a eventy. V herních enginech nás obzvlášť bude zajímat Tick, volaný každý časový krok simulace.

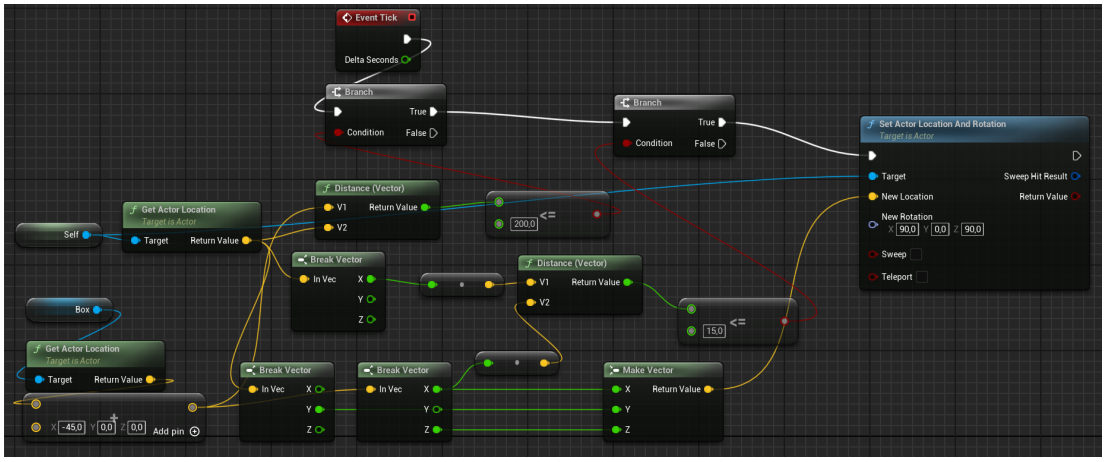
V Blueprintech jsem vytvořil jednu rodičovskou třídu jménem NativeDialIndicator, od které dědí všechny moduly (zpětně jsem chtěl jméno změnit, ale potřebný refactoring vždy vypnul Unreal Engine). Třída má Static Mesh jako svůj Root Component. Tento Mesh je zpravidla model modulu. Dědění mi umožnilo například neopakovat kód implementující přicvakávání modulů k tělu syntezátoru. V obrázku 4.1 můžete vidět, že při začátku simulace si tato třída najde objekt třídy Static Mesh Actor s Tagem 1 (tak je označen box pro moduly). Tato funkce se spustí i v každém dítěti této třídy a všechny moduly mají tedy od začátku simulace ukazatel na box, ke kterému se přichytávají. Je důležité tuto funkci volat pouze na začátku. Při hledání objektu s Tagem 1 musí engine projít všechny objekty třídy Static Mesh Actor, kterých je v mé scéně mnoho. Je to tedy časově náročná operace.



Obrázek 4.1. Ukázka funkce volané na začátku simulace v rodičovské třídě modulů.

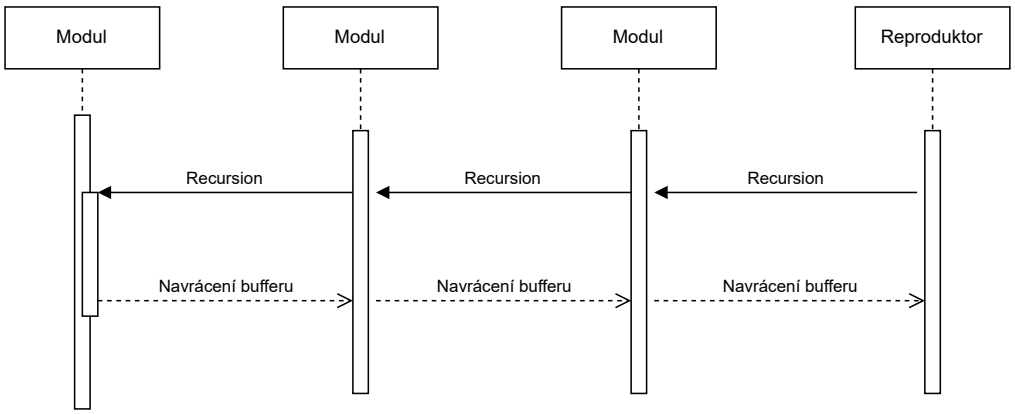
Event graph, tedy hlavní Blueprintový kód všech modulů, má vždy podobnou strukturu. Je ovládán takzvanými Eventy. Eventy jsou funkce spuštěné ve chvíli, kdy simulace zaznamená specifickou situaci a spustí kód v nich definovaný. V Event Graphu mých modulů se jedná o Event Tick, On Child Grip, On Child Grip Release a On BeginPlay. Následují funkce zprostředkované Blueprinty, které všechny moduly sdílí.

V dalším obrázku 4.2 je vidět zmíněný kód, ve kterém si třída zkontroluje své souřadnice a rotaci pomocí Get Actor Location s argumentem Self a v případě, že je dostatečně blízko (funkce Distance) k boxu, přichytí se k jeho straně a narovná patřičně rotaci.



Obrázek 4.2. Ukázka funkce volané každý Tick simulace v rodičovské třídě modulů.

Snad nejdůležitějším Blueprintovým kódem je systém, kterým si moduly přeposílají audio buffery. Skutečný modulární syntezátor šíří napětí svými kabely a v reakci na toto napětí moduly vysílají zase jiné napětí dál. V mém systému je přístup přesně opačný. Na začátku algoritmu je koncový reproduktor. Podívá se zda-li má kabel ve svém vstupu a zda-li je druhá část kabelu zapojena zase jinam. V kódu je tato funkce implementována veřejně viditelnou proměnnou PrevComp typu Actor, kterou modulu inicializuje kabel pouze v případě, že je korektně zapojen. V případě, že podmínka platí, pošle každý Tick zprávu připojenému modulu s žádostí o audio buffer, který by mohl přehrát. Tato zpráva je implementována veřejnou funkcí (ostatní třídy ji mohou vidět a volat) Recursion definovanou v rodičovské třídě všech modulů s návratovou hodnotou Buffer typu Array of Float velikosti 1024*8*2 (dále vysvětleno v části o C++ ??). Ve většině případů pak modul, na kterém reproduktor tuto funkci volá, zavolá identickou funkci na moduly zapojené v jeho vstupech, získá od nich buffer, upraví ho svými funkcemi ve svém C++ kódu a upravený buffer posílá reproduktoru. Využívám tedy tzv. spojový seznam. V případě, že v systému vytváří prvotní signál VCO, bude jeho funkce Recursion volána nejpozději. Funkce Recursion modulů budou předvedeny v sekci 4.3. Diagram 4.3 přestavuje tuto interakci.



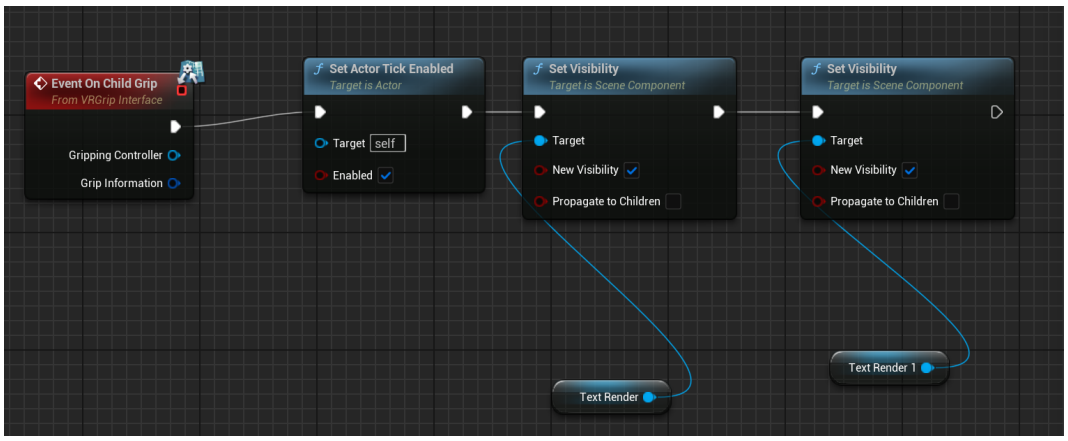
Obrázek 4.3. Představení přeposílání bufferu mezi moduly (šipky znázorňují kabely).

Sdílené Child Components modulů jsou texty třídy Text Render Component a knoflíky třídy VRDial Component. Dále mají moduly mnoho Static Mesh Components, které z největší části kompletují model modulu. Několik těchto komponentů nemá geometrii a jsou na pozici vstupů a výstupů. Tyto komponenty mají přiřazené Tagy využívané při propojování modulů a přichytávání kabelů na správná místa. V tabulce můžete vidět rozřazení Tagů.

Vstup	Výstup	Modulace
14		
21		
22		
23		
	13	
		15

Tabulka 4.1. Seznam Tagů.

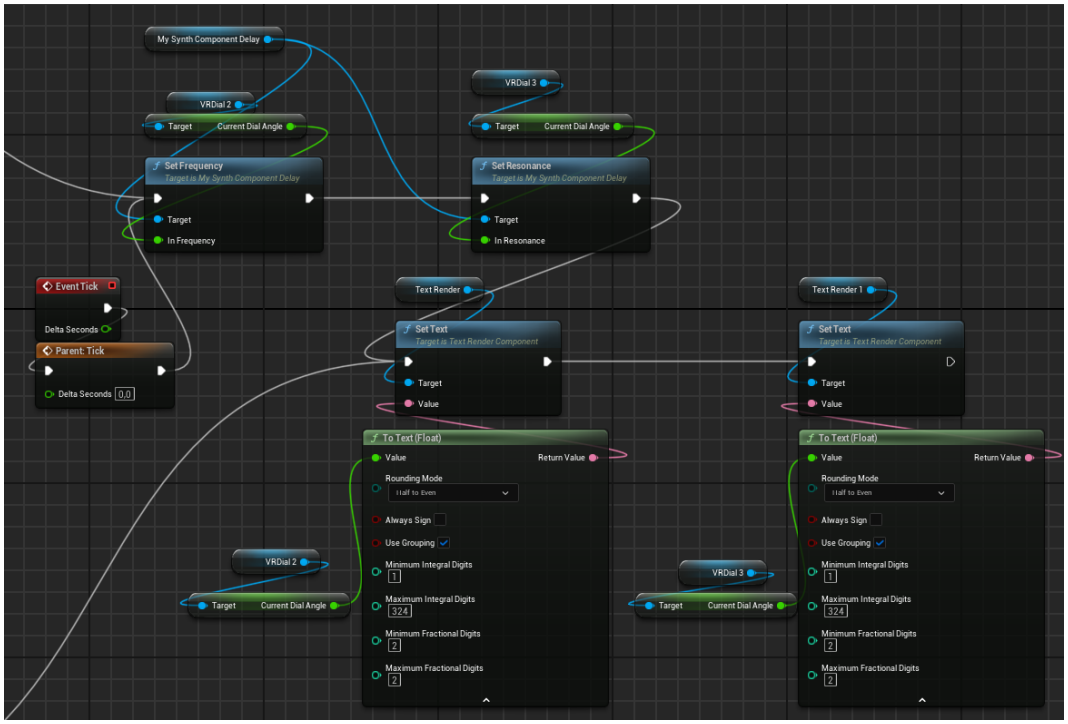
Kód zprovoznění knoflíku a zpětné vazby v podobě textu nad nimi se skládá z dvou částí, které můžete vidět v obrázcích 4.4 a 4.5.



Obrázek 4.4. Kód spuštěný v moment, kdy hráč chytí modul do ruky (Event On Child Grip), nastaví viditelnost textu na nepravdu.

Text nad knoflíky je viditelný na začátku simulace, aby uživatel věděl o ovladatelných částech syntezátoru. V průběhu ovládání ale pokaždé, když uživatel knoflík pustí, zmizí

a objeví se zas v moment, kdy uživatel zpětně knoflík uchytí. Dostává tedy zpětnou vazbu, že knoflík drží a může měnit jeho hodnotu.



Obrázek 4.5. Kód spuštěný každý krok simulace (Tick) nastaví viditelnost textu na pravdu a předá data textu.

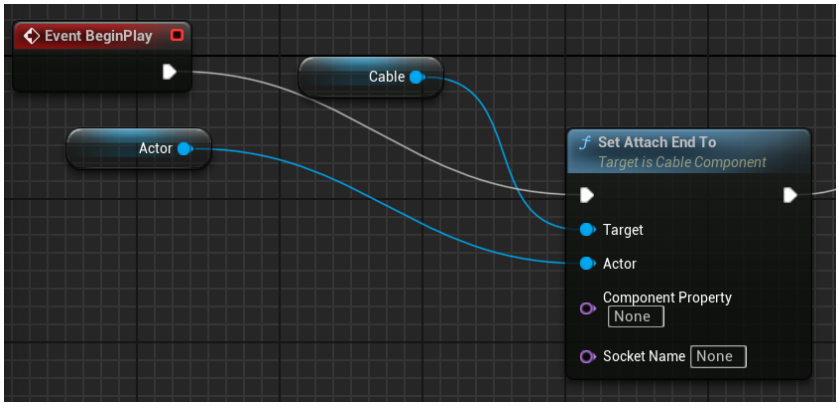
V moment, kdy uživatel knoflík drží, spustí se jeho Tick event následně volaný každý krok. V eventu se hledá hodnota Current Dial Angle, přeformátuje se na text a předá textovému poli. V modulech, které vyžadují jiná data než čistou rotaci, se zde také rotace mění na požadovaná data (např. 8' u VCO).

Vidět je i oranžový Parent Tick box. Ten zajišťuje, že se Tick funkce definovná v rodičovské třídě (základní modul) zavolá před následně definovanými příkazy.

Důležité proměnné rodičovské třídy jsou ukazatel na tělo syntezátoru (ukazatel typu Actor), ukazatel na až čtyři předchozí a pouze jeden následující modul (třídy Native Dial Indicator) a ukazatel specifický pro předchozí modul dodávající modulaci (třídy Native Dial Indicator).

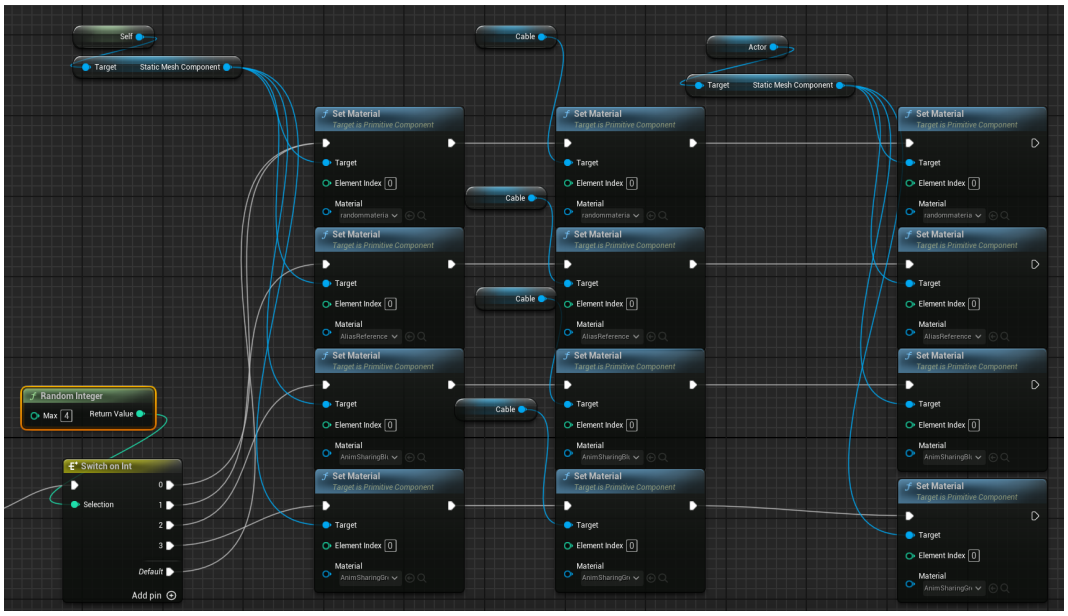
Vyjma modulů jsou další důležitou součástí simulace kabely. Propojení modulů mohlo být realizováno mnoha dalšími způsoby. V další několika odstavcích se pokusím nastínit svůj přístup.

Každý kabel je vytvořen dvěma objekty třídy StartCable a EndCable. Hned při spuštění simulace si dvojice těchto objektů nastaví svou proměnou typu Actor na druhý konec (může se na něj tedy odkazovat). StartCable i EndCable dědí od třídy Grippable Static Mesh Actor (třída dodaná VRE pluginem) dodávající možnosti uchycení do virtuálních rukou. Mají StaticMeshComponent s modelem kužele jako svůj Root Component a StartCable navíc obsahuje Cable Component. Cable Component se jedním koncem drží Root Componentu objektu StartCable a druhý konec se pomocí funkce Set Attach End To nastaví na příhodný EndCable (pomocí proměnné nastavené na začátku simulace).



Obrázek 4.6. Nastavení druhého konce kabelu.

Fyzika prohnutí kabelu je zcela ovládána enginem. Cable Component má mnoho možných nastavení, ovlivňujících fyziku, především počet pevných segmentů (mezi segmenty jsou klouby, ve kterých se kabel prohýbá). Viditelný kabel ovšem nedodává žádnou funkcionalitu, ta je schována v Blueprintech tříd StartCable a EndCable. Pro začátek je kýžená možnost různých barev kabelu implementována následovným jednoduchým principem. Hned po nastavení konce kabelu si kabel vybere náhodně ze čtyř různých materiálů se čtyřmi barvami (duhová, růžová, modrá, zelená). Tento povrch nastaví svému Root Componentu (kužel), Cable Componentu i Root Componentu EndCable jemu přiřazenému pro konzistentní barvu po celé délce kabelu.



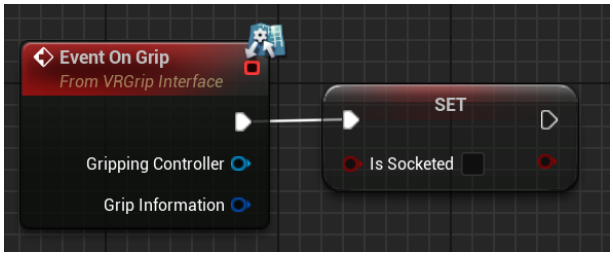
Obrázek 4.7. Nastavení náhodného materiálu pro kabel.

Důležité atributy tříd StartCable a EndCable můžete vidět v diagramu 4.8.

Cable
Cable: Actor
Native Dial Indicator: MyComponentStart
Bool: IsSocketed
Scene Component: Socket
Name: SockNumber

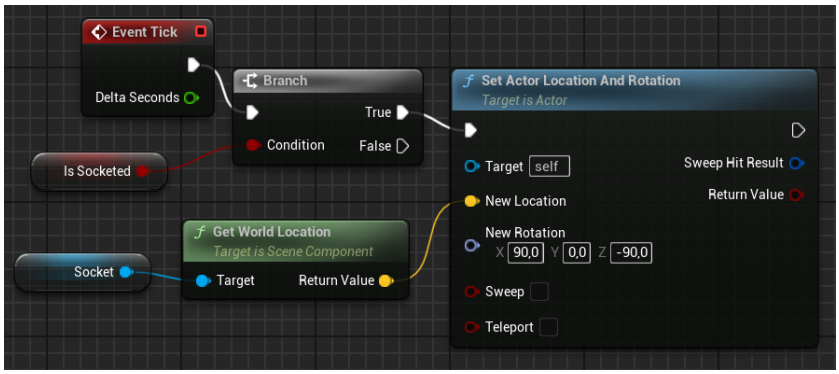
Obrázek 4.8. Atributy tříd StartCable a EndCable.

Princip propojení modulů za pomocí kabelu funguje v základu z obou stran analogicky.



Obrázek 4.9. Demostrace eventu On Grip.

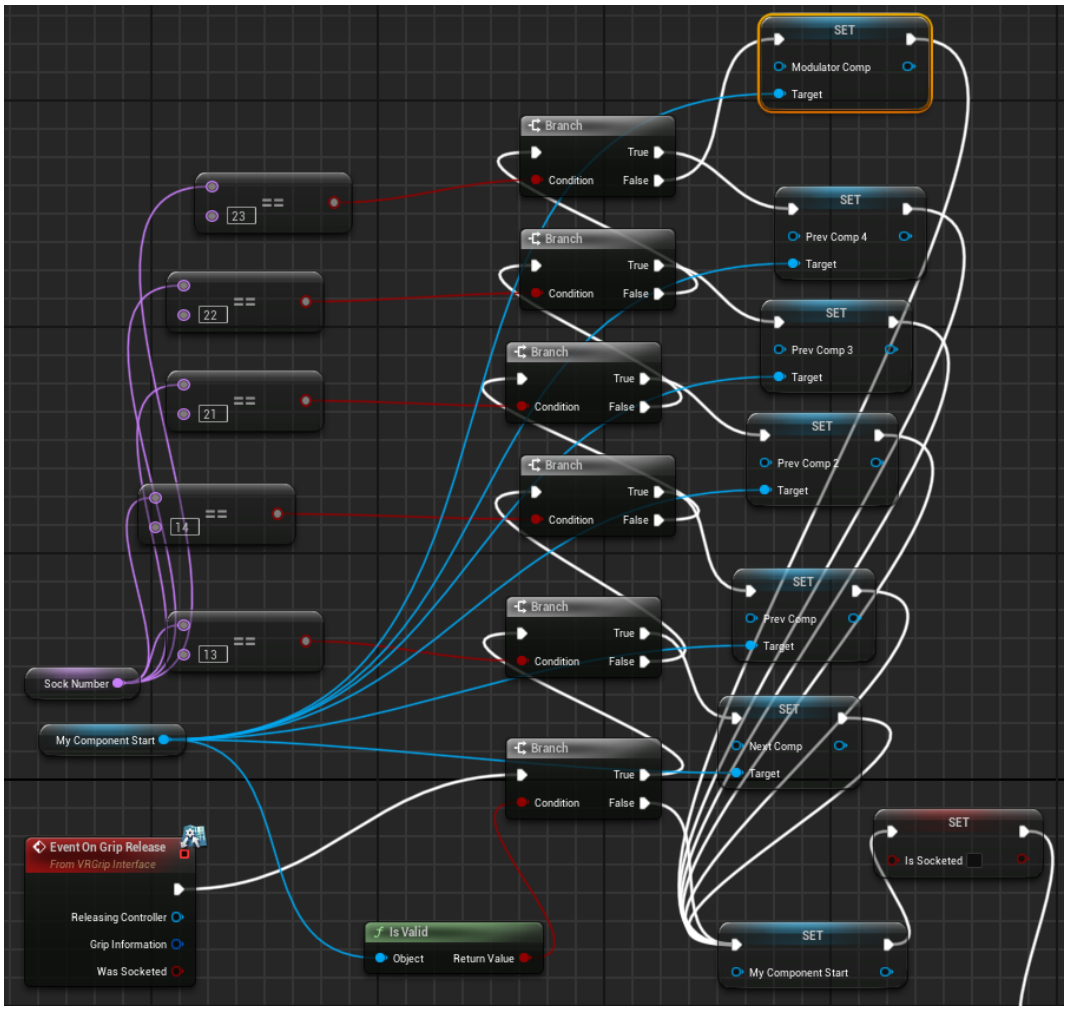
Každý Tick event si třída pomocí Switch boxu zkontroluje proměnnou IsSocketed a pokud je pravdivá, nastaví svou pozici (Set Actor Location And Rotation) na pozici proměnné Socket. Díky tomu se upraví poloha kabelu při pohybu s připojeným modulem.



Obrázek 4.10. Aktualizace pozice kabelu pro případ pohybu modulu.

Nejdůležitější funkce následují Event On Grip Release. Pokud byl náš konec už někam připojen (má inicializovanou proměnou MyComponentStart respektive MyComponent) musí se rozhodnout, co bude dělat v závislosti na SockNumber. SockNumber odpovídá Tagu u Child Component modulu, na který je kabel připojený. Pokud byl Tag 13 musí kabel nastavit PrevComp proměnnou svému MyComponentStart (MyComponent) na Null. Pokud uživatel, ale tento kabel pustil, znamená to, že s ním musel také pohnout. Je tedy vhodné výstup modulu odpojit. Pokud s ním pohnul a pustil ho zpět na stejné místo, propojení se zpátky nastaví v následujícím kódu. Podobně pokud byl náš SockNumber 14, 21, 22 nebo 23 musí se nastavit PrevComp (PrevComp2, PrevComp3, PrevComp4) našeho MyComponentStart na Null. Nakonec pokud ani jedna z

předchozích podmínek neprošla víme, že Tag, na který byl kabel připojen musel být 15 a nastavíme tedy ModulatorComp našeho MyComponentStart na Null. Vždy nastane pouze jedna (příp. žádná) z předchozích situací. Vzhledem k tomu, že předpokládáme, že po puštění kabelu se kabel od modulu odpojil nastaví se i náš MyComponentStart a IsSocketed (MyComponent) na Null a kabel tím ztratí odkaz na svůj předchozí modul syntezátoru. Tento kód můžete vidět v obrázku 4.11.

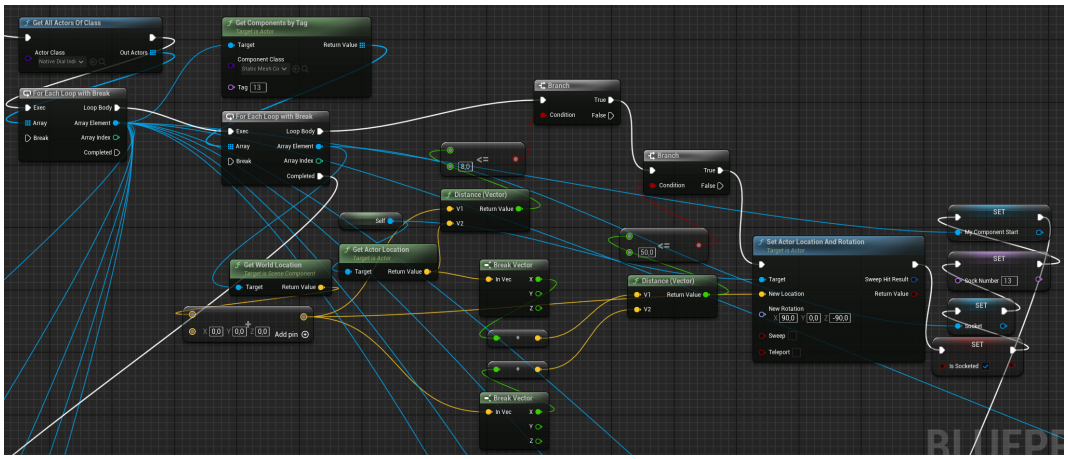


Obrázek 4.11. Odpojení kabelu od modulu syntezátoru po eventu On Grip Release.

Po tomto odpojení následuje kód implementující přicvakávání kabelů do vstupů a výstupů. Tato funkcionalita je znázorněna v pseudokódu 4.12.

```
Pro každý modul
    Pro každou komponentu modulu
        Pro každou komponentu modulu s Tagem 13
            Spočítej vzdálenost kabelu od komponenty
                Pokud je vzdálenost menší než určitá mez
                    Přicvakni kabel k této komponentě
                    Nastav kabelu IsSocketed na pravdu
                    Nastav kabelu Socket komponentu modulu
                    Nastav kabelu SocketNumber na příhodný Tag
                    Nastav kabelu MyComponentStart na modul
                    Přeskoč všechny další cykly
```

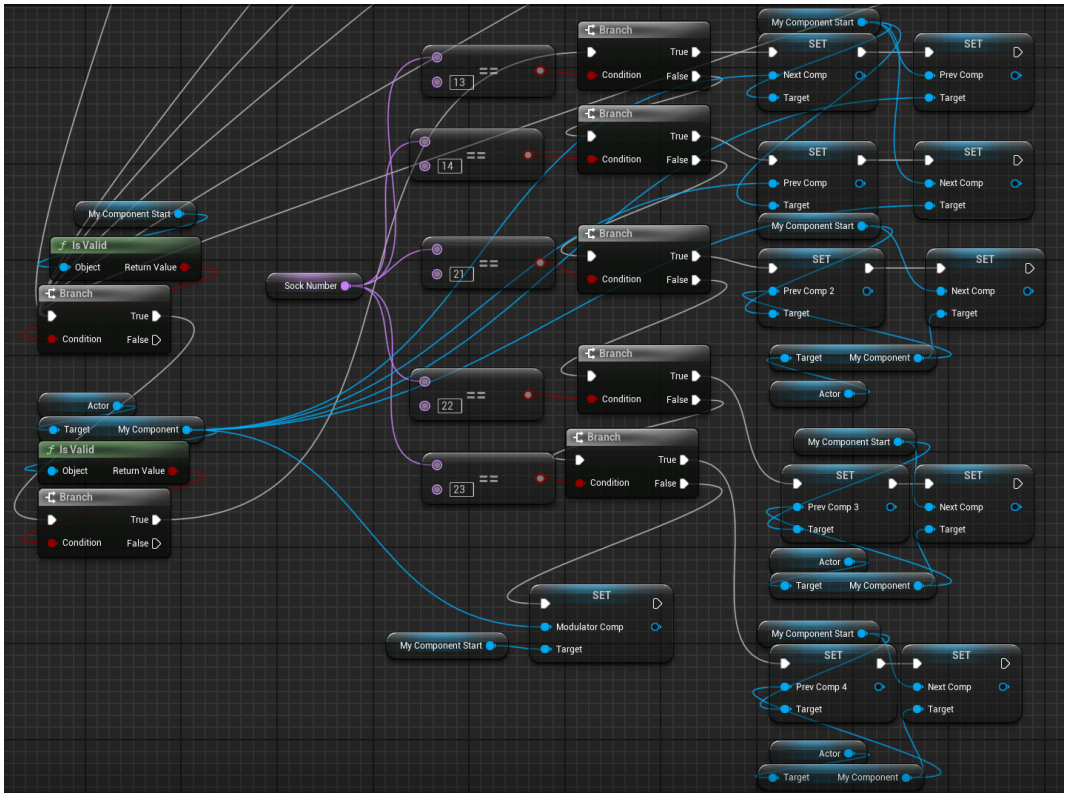
```
        Pro každou komponentu modulu s Tagem 14
            Stejně jako u Tagu 13
        Pro každou komponentu modulu s Tagem 15
            Stejně jako u Tagu 13
        Pro každou komponentu modulu s Tagem 21
            Stejně jako u Tagu 13
        Pro každou komponentu modulu s Tagem 22
            Stejně jako u Tagu 13
        Pro každou komponentu modulu s Tagem 23
            Stejně jako u Tagu 13
```



Obrázek 4.12. Přicvaknutí kabelu k blízkému vstupu nebo výstupu.

Kód eventu On Grip Release pokračuje. V této fázi je rozhodnuto, zda-li se konec kabelu přicvakl nebo jestli je vypuštěn do prostoru. Tato změna se musí propagovat i do modulu příslušného opačnému konci kabelu. Dále musíme brát v potaz, že pokud se přicvakl, má už nastavený IsSocketed, Socket, SockNumber i MyComponentStart na pravdivé hodnoty (pokud nepřicvakl jsou Null, popř. False). V případě, že náš MyComponentStart není Null (zjistíme znovu pomocí funkce Is Valid s argumentem MyComponentStart a boxu Branch) a zároveň není Null ani MyComponent opačného konce, event pokračuje (v opačném případě zde event končí).

Znovu se kontrolují SockNumber a v závislosti na nich se rozhoduje, který ze dvou propojených modulů si nastavuje PrevComp (PrevComp2, PrevComp3, PrevComp4) a který NextComp. Speciální případ je nastavení ModulatorComp. Vzhledem k tomu, že vypsání všech kombinací není nutné, můžete si ho prohlédnou na obrázku 4.13 a zde popíši jen případ, kdy náš SockNumber je roven 13. Náš MyComponentStart si nastavuje NextComp na MyComponent od EndCable a naopak MyComponent si nastavuje MyComponentStart na PrevComp. Jsme tedy v řetězci blíž ke generátoru bufferu a naše změny na posílaný buffer budou realizovány dříve, než změny, které udělá MyComponent.



Obrázek 4.13. Propagace zprávy o nově připojeném modulu na oba konce kabelu.

Třída EndCable obsahuje analogický systém. Oba konce kabelu se tedy chovají identicky.

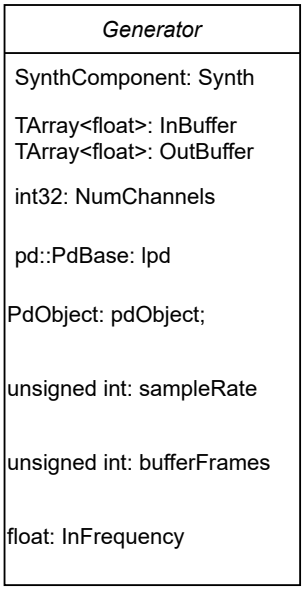
4.2.2 C++ a LibPD

Unreal Engine dovoluje kód tříd definovat pomocí Blueprintů a C++. Předchozí sekce se zabývala Blueprintovým kódem. Ten zajišťuje především komunikaci různých objektů v simulaci. C++ kód naopak v tomto projektu pracuje v každé třídě jen se svou vlastní třídou a s buffery, které přijímá. Všechny moduly mají třídu dědící od SynthComponent, její rozvržení je tedy stejné a mění se pouze proměnné a implementace funkcí. C++ kód, o kterém budu mluvit, je implementace této komponenty (celá třída modulu tedy nemá C++ kód).

Kód se skládá pokaždé z dvou tříd. Jedné dědící od třídy SynthComponent a druhé dědící od třídy SoundGenerator. SynthComponent třída si ve funkci CreateSoundGenerator vytvoří svůj generátor a ten komunikuje s knihovnou LibPD. Tento generátor poskytuje hlavní třídě buffery, které ona přeposílá do Blueprintového kódu. Důležité jsou specifikátory, které mohou mít funkce a proměnné v C++ kódu Unreal Enginu. Definují, které tyto položky jsou viditelné a volatelné z Blueprintů. Jsou to zejména BlueprintCallable u funkcí a EditAnywhere u proměnných.

SynthComponent má téměř vždy funkci, která předá buffer z jejího argumentu do generátoru, a funkci, která buffer naopak navrací. Dále má často BlueprintCallable funkce zprostředkující přeposílání hodnot z knoflíku do generátoru (ten je propaguje do LibPD). Dále má ukazatel na svůj generátor.

Proměnné Generátoru můžete vidět v diagramu 4.14.



Obrázek 4.14. Proměnné generátoru.

Při vytvoření generátoru ve funkci CreateSoundGenerator dostane generátor jako jeden z pěti argumentů ukazatel na SynthComponent, který ho vytvořil. V konstruktoru si nastaví své proměnné a pokusí se inicializovat LibPD s počtem vstupních a výstupních kanálů a s vzorkovací frekvencí. Dále nastaví pro LibPD posluchač pdObject. LibPD se pokusí otevřít Pure Data patch z cesty a nakonec se inicializují potřebné buffery na správné velikosti a zaplní se nulami. Následuje ukázka jednoho takového konstruktoru, specificky pro modul VCO.

```
FLPDGenerator::FLPDGenerator(int32 InSampleRate, int32 InNumChannels,
int32 InFrequency, float InVolume, UMySynthComponent* s)
   : NumChannels(InNumChannels){
   this->sampleRate = InSampleRate;
   this->bufferFrames = 128;
   this->NumChannels = 2;
   this->InFrequency = 0;
   this->Range = 263.111;
   this->synth = s;
   if (!lpd.init(0, 2, 44100)) {
      UE_LOG(LogTemp, Warning, TEXT("Could not init pd"));
   }
   lpd.setReceiver(&pdObject);
   lpd.subscribe("cursor");
   lpd.computeAudio(true);
   pd::Patch patch = lpd.openPatch("test.pd",
        "C:\\Users\\onder\\Downloads
        \\VRExpPluginExample-master\\VRExpPluginExample-master
        \\Content\\");
   lpd.sendFloat("FromCpp", 263.111);
   int block_size = lpd.blockSize();
   int ticks = 2;
   int channels = 2;
   this->synth->BufferToFilter.Init(0, 1024*8);
```

```
    inputBuffer.Init(0, 1024*8);
    outputBuffer.Init(0, 1024*8);}
```

Komunikace s LibPD se provádí pomocí volání různých metod nad instancí LibPD. Pokud bych chtěl do patche poslat float představující výšku tónu v hertzích v patchi vytvořím proměnnou FromCpp a v C++ zavolám metodu sendFloat s dvěma argumenty. První je jméno cílené proměnné a druhý hodnota, kterou chci poslat. Příkladem může být nastavení hranice dolní propusti modulu VCF.

```
void UMyMySynthComponentFilter::SetFrequency(float InFrequency){
   if (LPDGenerator.IsValid()){
      FLPDGeneratorFilter* ToneGen =
        static_cast<FLPDGeneratorFilter*>(LPDGenerator.Get());
      ToneGen->SetFrequency(InFrequency);
      ToneGen->InFrequency = InFrequency;}}

void FLPDGeneratorFilter::SetFrequency(float Frequency){
   SynthCommand([this, Frequency](){
         this->InFrequency = Frequency;});
   float f = Frequency / 360.0;
   f = f * 20000.0;
   lpd.sendFloat("FromCppf", f);}
```

Jak můžete vidět SetFrequency třídy SynthComponent pouze předává svůj argument stejnojmenné funkci v generátoru a další funkce neplní. Naopak vnitřní SetFrequency generátoru formátuje vstupní argument a přeposílá do LibPD. Rotace knoflíku se počítá v rozmezí 0 - 360 a my ji mapujeme na celé slyšitelné spektrum. Tento systém komunikace se s dalšími proměnnými a dalšími moduly opakuje, a to s pouze menšími úpravami.

Zajímavý kus kódu můžeme najít při zpracovávání příchozích a odchozích bufferů. Ačkoliv operujeme s více kanálovým signálem, LibPD od nás dostává pouze jednu řadu čísel. Řešení přináší prokládání. První hodnota prvního kanálu dostává index jedna, první hodnota druhého kanálu dostává index dva a tak dále dokud nedojde na všechny kanály a od druhé hodnoty se princip opakuje. Zajímavé je, že pokud máme více zdrojů, levý a pravý kanál každého zdroje musí být hned za sebou. I z tohoto důvodu je velice důležité inicializovat LibPD se správným počtem kanálů. Více kanálů používáme z několika důvodů. Oproti tradičním modulárním syntezátorům dostáváme výhodu stereofonního signálu a umožňují nám do LibPD posílat audio buffer a buffer modulačních dat zároveň nebo třeba více audio bufferů.

Následuje ukázka takového zpracování dat do bufferu pro LibPD v modulu mixeru. LibPD dostává osm kanálů ze čtyř zdrojů a tato funkce poskytuje sedmý a osmý (poslední zdroj s levým a pravým kanálem). Iteruje přes buffer z argumentu (dodaný pomocí Blueprintů ve funkci Recursion) a hodnoty předává do bufferu pro LibPD, počínaje na indexu šest.

```
void UMySynthComponentMixer::LOLsBufferFromOsc4(TArray<float>
InBufferFromOsc){
   if (InBufferFromOsc.Num() == 0) {
      UE_LOG(LogTemp, Warning, TEXT("InBufferFromOsc4 is empty"));
      return;}
   int j = 0;
   for (int i = 6; i < 1024 * 8 * 4; i += 8) {
```

```
        this->BufferFromOsc[i] = InBufferFromOsc[j];
        this->BufferFromOsc[i + 1] = InBufferFromOsc[j + 1];
        j += 2;}}
```

Metoda posílající buffer z naší instance LibPD do Blueprintů je naopak velice jednoduchá a s bufferem přijatým od LibPD nic nedělá. Další opakující se funkce jsou funkce, které resetují buffer zpátky na nulové hodnoty v případě, že se kabel ze vstupu odpojil.

Poslední a snad nejdůležitější funkce je OnGenerateAudio ve třídě generátorů. Právě v ní se volá processFloat na LibPD s třemi argumenty. První definuje počet ticků. Tick označuje jednotku měření určující latenci zpracování zvuku. Definujeme, kolik ticků za jeden buffer LibPD zpracuje (jeden tick má velikost blocksize*počet kanálů, blocksize definuje LibPD jako 64). Menší počet ticků zrychlí zpracování bufferu, ale může způsobit výpadky zvuku. Druhý argument je ukazatel na vstupní buffer a poslední ukazatel na výstupní buffer, do kterého bude LibPD zapisovat. Na konci této funkce se výstupní buffer připraví podle prokládacího systému do bufferu odesílaného do Blueprintu.

```
int32 FLPDGenerator::OnGenerateAudio(float* OutAudio, int32 NumSamples){
   if (this->outputBuffer.IsEmpty()) {
      this->outputBuffer.Init(0, 1024*8);
      UE_LOG(LogTemp, Warning, TEXT("outputBuffer is empty"));}
   if (this->inputBuffer.IsEmpty()) {
      this->inputBuffer.Init(0, 1024*8);
      UE_LOG(LogTemp, Warning, TEXT("inputBuffer is empty"));}
   lpd.processFloat(64, &(inputBuffer[0]), &(outputBuffer[0]));
   this->NumChannels = 2;
   int32 NumFrames = NumSamples / this->NumChannels;
   int32 SampleIndex = 0;
      for (int32 FrameIndex = 0; FrameIndex < NumFrames; ++FrameIndex){
         for (int32 Channel = 0; Channel < NumChannels; ++Channel){
            this->synth->BufferToFilter[SampleIndex] =
                outputBuffer[SampleIndex];
            SampleIndex++;   }}
   return NumSamples;}
```

4.2.3 Pure Data

Toto je sekce přestavující společné vzory mezi patchi. Téměř všechny moduly používají v Pure Datech boxy adc~ a dac~. ~ značí, že vstup boxu bude v podobě signálu. Adc je konvertor analogového signálu na digitální a získává data ze vstupního bufferu (druhý argument metody processFloat v C++). Boxem adc~ se zpravidla začíná a z něj se postupně mohou připojit jednotlivé kanály na různé upravující funkce. Tyto funkce se nakonec spojí do vstupu boxu dac~. Ten data zapisuje do výstupního bufferu (třetí argument metody processFloat v C++).

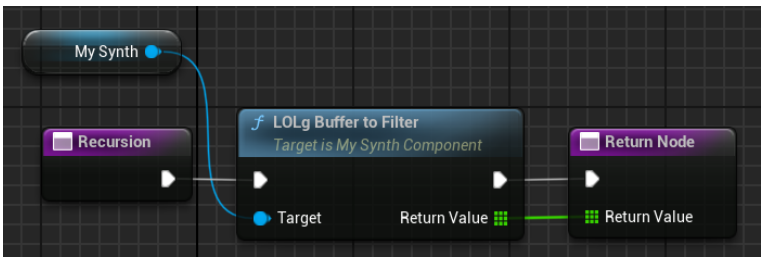
Často používané metody Pure Dat jsou například *~, +~, s~ a r~. Právě boxy r (recieve) společně se správným jménem přijímají zprávy z C++. *~ má dva vstupy. Časté využití této metody posílá do levého vstupu data zvukového signálu a do pravého číslo od 0 do 1, které upravuje hlasitost daného signálu. Další důležité funkce se neopakují a budou popsány v následující sekci.

4.3 Jednotlivé moduly syntezátoru

V této sekci se podíváme na moduly separátně. Popsány budou metody specifické pro jednotlivé moduly a jejich Pure Data patche. Všechny moduly dostaly vysvětlivky i uvnitř aplikace, viditelné v moment držení daného modulu.

4.3.1 VCO

Začneme s funkcí Recursion, jelikož u té jsme ještě neviděli příklad a v tomto modulu je snad nejjednodušší.



Obrázek 4.15. Příklad metody Recursion z modulu VCO.

Zajímavý může být způsob ovládání druhého knoflíku. Naším cílem je dát uživateli možnost volného pohybu s knoflíkem, ale přicvaknout hodnotu na jednu z šesti možností po upuštění knoflíku. Jedná se o knoflík vybírající frekvenční rozsah.

```
Pokud uživatel pustí knoflík:
    Pro každý rozsah nejbližších rotací kolem hodnot:
        Pokud je aktuální rotace v tomto rozsahu:
            Zapiš tuto rotaci
            Zapiš odpovídající výšku (např. 16')
            Nastav text nad knoflíkem na tuto výšku
            Nastav rotaci knoflíku na odpovídající rotaci
```

Definujeme řadu čísel odpovídajících rotacím, na které se chceme přichytit. Po vypočítání nejbližší hodnoty se tato rotace zapíše a text nad knoflíkem se nastaví na správnou hodnotu, stejně tak nastavíme i rotaci knoflíku na tuto rotaci.

```
void FLPDGenerator::SetRange(float RRange){
   if (std::abs(RRange - 87.0) <= 5.0) {
      this->Range = 263.111/50.0;}
   else if (std::abs(RRange - 115.0) <= 5.0) {
      this->Range = 65.4;}
   else if(std::abs(RRange - 142.0) <= 5.0) {
      this->Range = 130.8;}
   else if (std::abs(RRange - 165.0) <= 5.0) {
      this->Range = 261.6;}
   else if (std::abs(RRange - 195.0) <= 5.0) {
      this->Range = 523.2;}
   else if (std::abs(RRange - 217.0) <= 5.0) {
      this->Range = 1046.5;}
   else {
      this->Range = 261.6;}
   this->SetFrequency(this->InFrequency);
}
```

Dále rotaci přeposíláme do C++ a v ní ji přepočítáváme na skutečnou výšku v hertzích. Zajímavá je metoda generátoru v C++ upravující výšku prvním knoflíkem. Nejen, že musí brát v potaz vybraný rozsah, ale nulová hodnota je na dvanácti hodinách. Rotace na levou stranu ubírá až jednu celou oktávu a naopak rotace doprava oktávu přidává.

```
void FLPDGenerator::SetFrequency(float Frequency)
{
   SynthCommand([this, Frequency]()
      {
      });
   this->InFrequency = Frequency;
   float oldf = this->Range;
   float newf = 0.0;
   if (Frequency >= 180.0) {
      Frequency -= 180.0;
      Frequency = Frequency / 180.0;
      Frequency = 1.0 - Frequency;
      Frequency = Frequency * oldf;
      newf = oldf - (Frequency/2);
   }
   else {
      Frequency = Frequency / 180.0;
      Frequency = oldf * Frequency;
      newf = oldf + (Frequency);
   }
   lpd.sendFloat("FromCpp", newf);
}
```

Modul oscilátoru získal oproti návrhu několik úprav. Přibyl mu vstup pro modulaci frekvence a knoflík na plynulé ovládání waveform. Na výběr má uživatel mezi sin, saw, square, triangular a přechody mezi nimi. Vstup modulace nepřijímá audio buffer, ale slouží pro nastavení ukazatele na tento modul pro dva možné vstupy (theremin a sequencer).

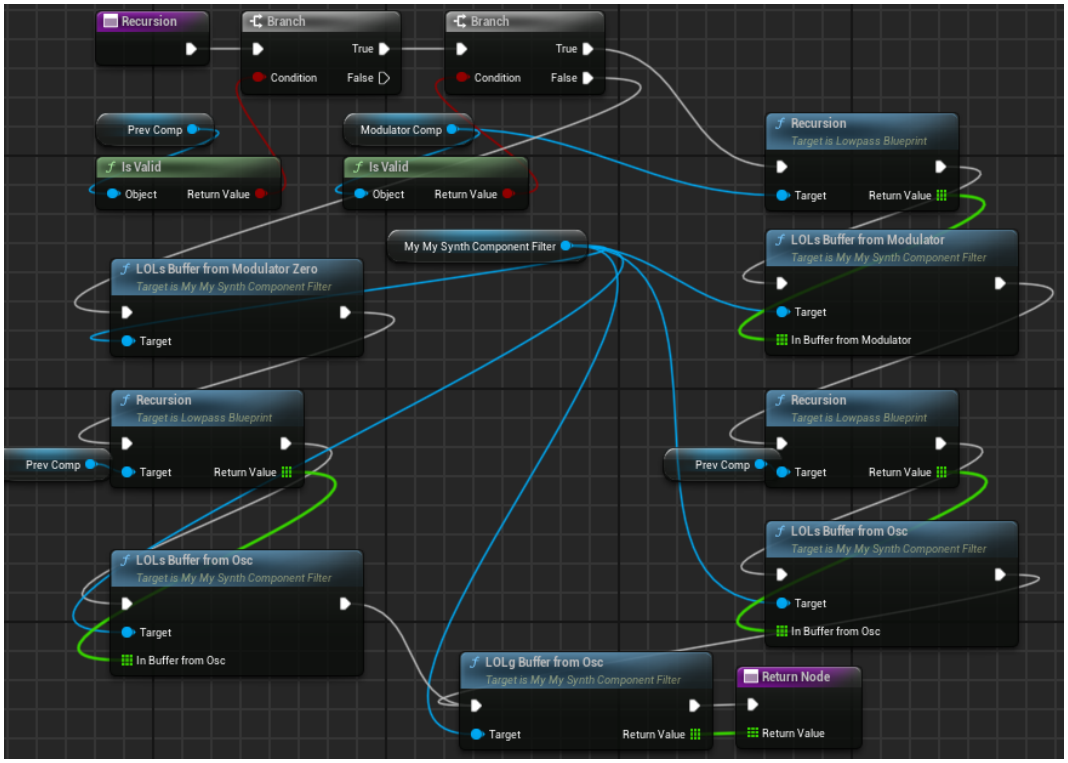
4.3.2 White noise

Tento modul je velice jednoduchý. Obsahuje pouze metody popsané ve společné části. Pure Data patch pro bílý šum je také velice jednoduchý, jak můžete vidět na obrázku 4.16.

Obrázek 4.16. Pure Data patch pro bíly šum.

4.3.3 VCF

V modulu VCF můžeme pozorovat zajímavou verzi funkce Recursion.

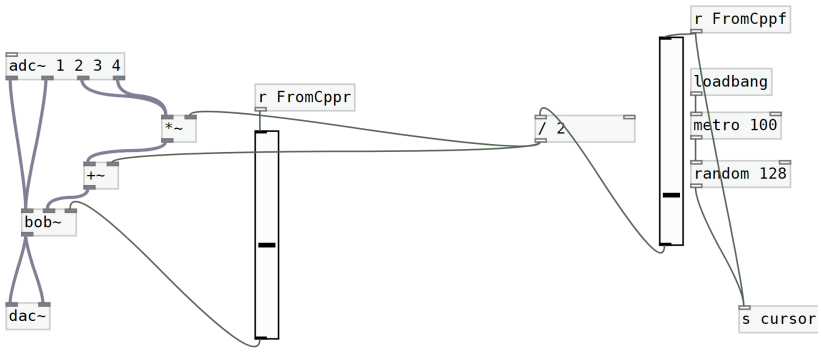


Obrázek 4.17. Recursion funkce modulu VCF.

Modul si zkontroluje, zda-li má na vstupu zdroje pro příchozí audio buffer a modulační buffer. Podle toho se rozhoduje, jestli pošle do Pure Data patche prázdné buffery nebo jednoduše přepošle příchozí. Nakonec v obou případech zavolá funkci pro získání odchozího bufferu a navrátí jej.

V Pure Data patchi VCF můžeme vidět dva posluchače pro ovládání parametrů propustě. Samotná filtrace probíhá v objektu bob (jméno po Bobovi Moogovi). Dostává audio buffer, hranici propusti FromCppf a resonanci FromCppr. Slidery jsou v patchi kvůli testování při tvorbě patche. Vidět můžeme implementaci modulace. Třetí a čtvrtý

kanál jsou násobeny polovinou frekvence hranice (přicházejí s rozsahem -1 až 1) a pak je k nim tato polovina přičtena. Signál se tedy pohybuje z poloviny hraniční frekvence směrem nahoru o polovinu hraniční frekvence (dosahuje tedy stanoveného maxima) a pak zpátky dolů až na nulovou hodnotu.



Obrázek 4.18. Pure Data patch modulu VCF.

4.3.4 VCA

Tento modul neimplementuje žádné změny k návrhu. V Pure Datech je velmi jednoduché schéma násobení signálu modulačním signálem a následně vstupní hodnotou FromCpp.

4.3.5 Mixer

Mixer nejdříve vynásobí každý ze čtyř vstupů odpovídající hodnotou jednoho ze čtyř knoflíků, následně je sečte a pošle na jeden svůj výstup. Implementačně zde bylo náročné správné prokládání kanálů.

4.3.6 ADSR

Generace křivky u ADSR je implementována přesně dle návrhu. Unikátní část toho modulu je jeho Pure Data patch. V Pure Datech se křivka ze získaných čtyř hodnot tvoří pomocí boxů message a vline~. Pack funkce dovolí message boxu přistupovat k vstupním hodnotám pomocí ukazatele. Message definuje, za jak dlouho se křivka dostane na jaké hodnoty. Vline~ vytvoří z křivky signál.

Důležitá část tohoto modulu, která nebyla zmíněna v návrhu, je signál pro vytvoření křivky. ADSR modul má jeden výstup pro křivku a modulační vstup právě pro tento signál. Podobným způsobem jako modulační vstup oscilátoru nepřijímá audio buffer, ale slouží jako propojení se sequencerem nebo thereminem, přijímá ADSR signál z sequenceru, který ho dokáže aktivovat.

4.3.7 Sequencer

Sequencer byl oproti návrhu výrazně změněn. Společně s thereminem jsou to jediné dva moduly bez Pure Data patche. Má pouze jeden výstup a nevychází z něj audio buffer. Naopak na moduly na výstupu volá funkce. Funguje tedy přesně opačně (stejně tak i theremin) oproti propojování normálních Pure Data modulů. Mimo knoflíky na ovládání jednotlivých kroků a rychlosti dostal také tlačítko na synchronizaci pro případ, kdy je ve scéně sekvencerů více. Má vnitřní třídu, která slouží jako hodiny s nastavitelným krokem. Vnější třída jí nastavuje čas mezi jednotlivými voláními eventu Tick a naopak

vnitřní třída z eventu Tick upozorňuje vnější třídu k dalšímu kroku. Každý krok má pouze jeden knoflík a implementace změny hlasitosti i výšky za použití sequenceru si tedy žádá dva synchronní sequencery.

4.3.8 Delay

Delay modul používá ve svém Pure Data patchi delwrite~ a delread~ funkce schopné přijmout signál a mezi sebou si ho s definovaným zpožděním přeposlat. Pomocí smyčky přeposílání vstupního signálu, ve které se při každém opakování signál zeslabí, je možné vytvořit příjemně znějící delay efekt. Pomáhají i dolní propustě dále měnící barvu opakovaného zvuku. Nakonec se signál spojí s čistým signálem a efekt je hotový. Bonusová funkcionalita je definice rozdílného času zpoždění pro levý a pravý kanál, tvořící příjemný stereofonní efekt.

4.3.9 Reverb

Reverb efekt je vytvořen spojením mnoha delayů. Oproti návrhu druhý knoflík neovládá barvu efektu, ale poměr mezi čistým a efektovaným signálem na výstupu. Uživatel má díky tomu možnost efektovat velice jemně.

4.3.10 Theremin

Theremin je implementován pomocí dvou tříd. Ačkoliv theremin a sequencer nemají Pure Data patch, dědí stejně jako všechny ostatní moduly z rodičovského modulu (z důvodu spolupráce s kabely). Bylo by velmi náročné přidávat modulu dva výstupy. Rozhodl jsem se tedy vytvořit dva moduly - jeden pro horizontální a druhý pro vertikální snímač. Mají své vlastní výstupy, ale jejich těla jsou spojena a pro uživatele se jeví jako jeden modul. Obě třídy na modul připojené k výstupu kontinuálně volají funkci předávající vzdálenost od snímače. Tato vzdálenost je matematicky upravena, aby odpovídala chování skutečných thereminů. Snímač snímá jen do určité vzdálenosti a následně přeposílá konstantní hodnotu blížící se nule. Hodnoty uvnitř této vzdálenosti se namapují mezi 0 a 1, vynásobí -1 a přičtou k 1, aby se při přibližování k snímači zvyšovaly. Násobí se a logaritmují logaritmem o základu 100 pro přesnější ovládání vysokých hodnot (vyšší noty mají k sobě frekvenčně blíž než noty nižší a hodnota 100 vychází z testování).

4.3.11 Distortion

Jelikož sequencer a theremin nemají Pure Data patch, rozhodl jsem se pro splnění zadání této práce k modulům přidat dva další - distortion a ring modulator. Distortion má vstup pro zvuk a výstup pro efektovaný zvuk. Dále má knoflík ovládající úroveň zkreslení.

Zkreslení nejdříve čistý signál, který nabývá hodnot na intervalu -1 až 1, násobí hodnotou z knoflíku. Amplituda signálu je tedy navýšena a zvuk je hlasitější. Dále bychom tradičně použili clip funkci, která signál navrátí zpátky na interval -1 a 1 a hodnoty, které se nacházely mimo tento interval, by se připnuly na maximální hodnotu 1 respektive -1. Pro více zajímavé zkreslení ale v tomto modulu implementuji tzv. wave folding. Křivka stoupající nad hodnotu 1 se zalomí a se stejným zrychlením začne klesat (analogicky se zápornými hodnotami). Takto se několikrát odrazí, dokud ji nezastaví funkce clip.

4.3.12 Ring modulator

Tento modul má dva vstupy, knoflík a jeden výstup. Signály ze vstupu se nejdříve sčítají, ale s otáčením knoflíku se postupně přestávají sčítat a začínají se násobit. Násobení oscilátoru s nízkou frekvencí s oscilátorem s frekvencí vysokou může tvořit zajímavé barvy.

4.4 Menu

V požadavcích pro tuto práci je generování modulů a kabelů a načítání a ukládání scény. Tyto interakce jsou společně s vypínáním zpřístupněny v menu implementovaném pomocí třídy WidgetSelectorUI a její třídy Widget. Widget definuje panel s tlačítky. Tlačítka jsou pomocí eventů stisknutí napojena na funkce s implementací představených akcí. Ve třídě WidgetSelectorUI se v každém Ticku nastavuje pozice a rotace Widgetu, aby odpovídala levé ruce hráče. Zbytek logiky je ve třídě hráče. Event stisknutí tlačítka X na levém ovladači vytvoří instanci třídy WidgetSelectorUI s pozicí před levým ovladačem a pravý ovladač na jednotlivá tlačítka může klikat (pokud na ně míří svým laserem).

Pro generování modulů stačí pozice a v případě kabelů přidání napojení StartCable na EndCable. Ukončení hry je jednoduché. Implementuje navíc pouze postupné ztmavnutí obrazovky. Komplikované je však načítání a ukládání scény.

4.4.1 Načítání a ukládání

Unreal Engine disponuje ukládacím systémem se základem ve třídě Save Game. Dovoluje Vám do této třídy přidat proměnné a dále je pomocí funkcí Create Save Game Object, Save Game to Slot a Load Game from Slot ukládat. Nedokáže však uložit automaticky celou scénu. Existují různé placené pluginy, které tyto funkce doplňují [1]. Save Game nedokáže ani ukládat instance Actorů a v mém případě tedy ani ukazatele kabelů na připojené moduly.

Vytvořil jsem tedy mnoho Structů uchovávajících všechny proměnné, pozice a rotace knoflíků mých tříd a ty jsem ukládal. Při načtení scény se všechny existující moduly a kabely smažou a podle uložených hodnot se na správných místech vytvoří předem uložené objekty. Vzhledem k různých počtům knoflíků a různým proměnným mých tříd je tato implementace velice rozsáhlá - nebylo možné jednoduše ukládat a načítat všechny objekty dědící od rodičovského modulu. Po vytvoření nových modulů se na všech kabelech zavolá funkce identická funkci spuštěné po upuštění kabelu a ty se tedy připnou na blízké vstupy a výstupy a správně nastaví modulům ukazatele. Uživatel má možnost uloženou scénu z paměti vymazat pomocí tlačítka Delete Save.

4.5 Tvorba scény

Scénu jsem tvořil pomocí Landscape a Foliage módů Unreal Enginu. Landscape mód dovoluje upravovat zem se sochařskými nástroji. Vytvořil jsem několik menších hor a jezero. Scéna pro uživatele se nachází v údolí mezi těmito scénami. Foliage mód si nejdříve vyžádá objekty stromů a květin a dále je vývojáři dovolí různými nástroji rozmísťovat po scéně. Kromě tohoto módu jsem prvotně využíval i Procedural Content Generation framework (PCG) přidaný s verzí 5.2. Dovoluje vývojáři v Blueprintech

[1] například `https://www.unrealengine.com/marketplace/en-US/product/simple-save-game`

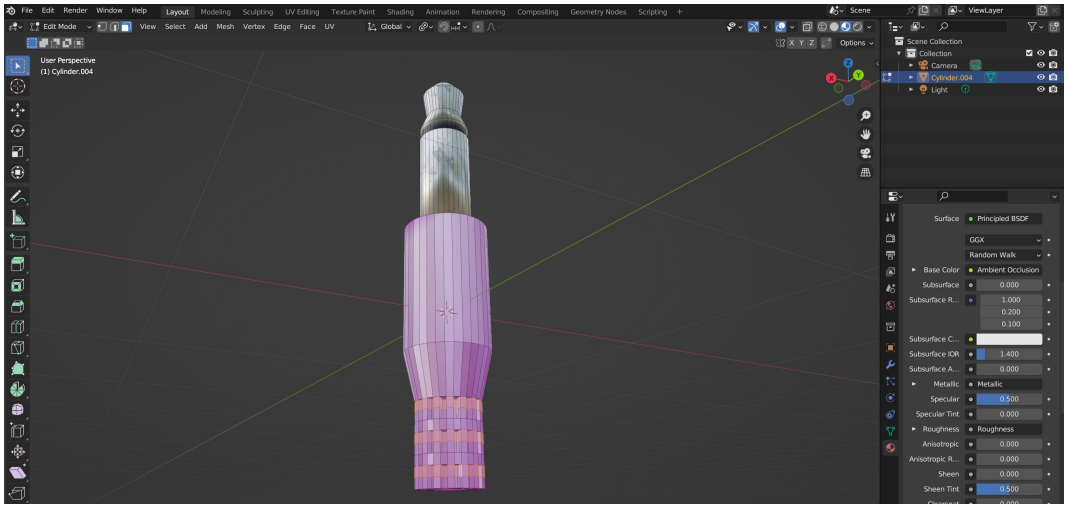
definovat pravidla pro procedurální umísťování objektů. Vzhledem k optimalizaci běhu aplikace ale nebylo žádoucí, aby objektů ve scéně bylo příliš, a ve výsledné aplikaci jsou tedy objekty všechny položeny manuálně.

Scénu jsem tvořil dvakrát. První představená scéna využívala objekty z Unreal Engine Marketplace a musel jsem je nahradit. V nynější verzi jsem objekty scény získal z různých zdrojů ovšem s licencí CC0. Importování těchto objektů bylo však časově náročné. Žádný formát nebyl enginem přijímán jednoduše a mnoho objektů nebylo možné importovat vůbec. U těch, u kterých se import podařil, jsem musel tvořit materiály ručně a opravovat transformace. Problém také tvořily normály. Při zapnutém backface culling jsem často viděl do stolů nebo jejich nohou i přes otáčení normál. Tento problém způsoboval neuzavřený povrch těchto objektů. Řešením se ukázala two sided možnost u materiálů v Unreal Enginu.

Všechny objekty, se kterými uživatel interaguje, jsem tvořil v programu Blender.

4.5.1 Tvorba objektů

V Blenderu jsem vytvořil stoly, moduly, kabely, theremin, reproduktor a tělo syntezátoru. Dále jsem Blender sporadicky používal k opravování importovaných objektů. Při tvorbě těchto objektů bylo obzvlášť důležité správně vytvořit realistické materiály s texturami pro normály, metalízu, bump mapy a další. Objekty jsou tvořeny ze základních tvarů (krychle, hranol, koule) a dále upravovány funkcemi v Edit módu Blenderu. Hlavní funkce v této tvorbě byly loop cut, knife, scale, move, rotate a extrude. Na obrázku 4.19 můžete vidět tvorbu jacku pro kabel.



Obrázek 4.19. Tvorba jacku v Blenderu.

Po vytvoření kompletních modulů se objevil další problém. Moduly jsou tvořeny několika objekty - knoflíky, plíšky u vstupů. Tyto objekty se do Unreal Enginu importovaly se špatným nastavením lokálních souřadnic - jejich lokální středy byly uprostřed modulu. To komplikovalo rotace knoflíku a upravování umístění vstupů. Nakonec jsem často musel moduly tvořit z těchto stavebních objektů znovu v Unreal Enginu.

Kapitola 5
Testování

Ve dnech 15.05.2024 a 16.05.2024 probíhalo v budovách ČVUT na Karlově náměstí a v Dejvicích kvalitativní testování s osmi testujícími. Testující byli vybíráni dle popisu cílové skupiny 3.1. Při výběru jsem oslovoval jednotlivce buď více zkušené s virtuální realitou nebo naopak s modulárními syntezátory. To z toho důvodu, že testery přesně odpovídající cílové skupině bylo kvůli jejímu úzkému zaměření velice náročné vyhledat.

5.1 Příprava

V přípravě testování jsem vytvořil dotazníky, navrhl úkol pro testující, finalizoval rozvržení pracovní stanice v simulaci a vysvětlivky kolem ní. Testující se objevili u prázdného těla syntezátoru pouze s reproduktorem a thereminem.

5.2 Průběh

Testující byli nejdříve obeznámeni s průběhem testování. Dále dostali vstupní dotazník A. Byl jim představen jednoduchý úkol:

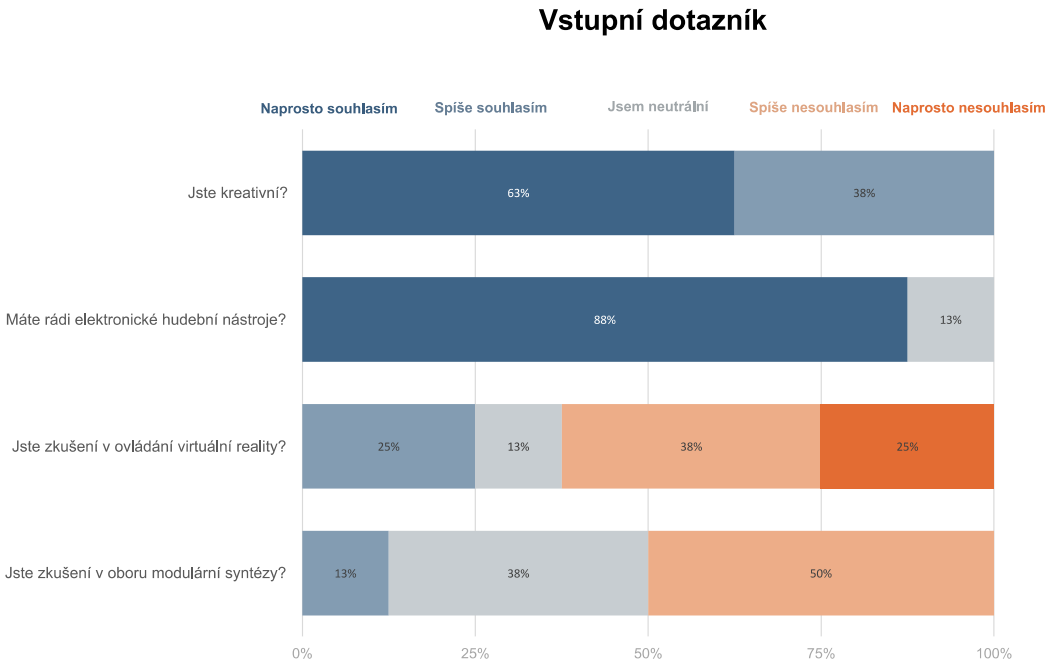
Vytvořte jednoduchou modulární sestavu s minimálně jedním generátorem zvuku (white noise nebo oscilátor), jedním dalším modulem, modulem efektu a reproduktorem. Dále se pokuste zvuk tohoto sestavení upravit pomocí knoflíků podle svého zalíbení. Soustavu uložte a aplikaci zavřete.

Testující byli požádání, aby popisovali nahlas, co dělají, co by chtěli dělat, co hledají, co jim vadí, co je naopak příjemné a co si bere jejich pozornost. Mohli se doptávat. Testujícím byl představen přibližný časový rozsah úkolu dvacet minut. Před vstupem do aplikace se seznámili s rozhraním ovladačů Oculus Quest Touch. V průběhu jsem si zapisoval poznámky, reagoval na připomínky testujících a celé testování bylo zaznamenáváno na statické video. Na konci testování dostali testující výstupní dotazník B.

5.3 Výsledky

Mezi jednotlivými testováními byly časové úseky, které jsem využil k upravování aplikace dle získané zpětné vazby. Většina testujících časový rozsah úkolu překročila, v jednom případě o téměř čtyřicet minut. Ne však z důvodu náročnosti úkolu, ale protože i po splnění vyjadřovali zájem o zkoumání dalších možností aplikace.

Následují výsledky vstupního dotazníku. Nejprve graf vytvořený z otázek využívajících Likertovu škálu 5.1.



Obrázek 5.1. Výsledky Likert části vstupního dotazníku.

Dále odpovědi na otevřené otázky 5.2.

Otázky	Tester 1	Tester 2	Tester 3	Tester 4
Máte vlastní modulární syntezátor?	Ano	Ne	Ne	Ne
Máte vlastní brýle pro virtuální realitu	Ne	Ne	Ne	Ne
Jak často používáte virtuální realitu?	1x za rok	1x za 5 let	Velmi vzácně	1x za rok
Jaký je váš oblíbený hudební žánr?	jazz, klasika	jazz, metal, prog	rap	alternativní rok
Pokud jste muzikant, na jaký hudební nástroj hrajete?	kytara	baskytara	kytara	baskytara, bicí
Otázky	**Tester 5**	**Tester 6**	**Tester 7**	**Tester 8**
Máte vlastní modulární syntezátor?	Ne	Ano	Ne	Ne
Máte vlastní brýle pro virtuální realitu	Ne	Ano	Ne	Ne
Jak často používáte virtuální realitu?	Nikdy	6x do roka	1x ročně	2x do roka
Jaký je váš oblíbený hudební žánr?	indie pop-rock	DnB, Deathcore	klasická hudba, electro funk	stoner - rock
Pokud jste muzikant, na jaký hudební nástroj hrajete?	kytara, bicí, klavír, zpěv	Kytara, klávesy	multiinstrumentalista	kytara

Obrázek 5.2. Výsledky otevřené části vstupního dotazníku.

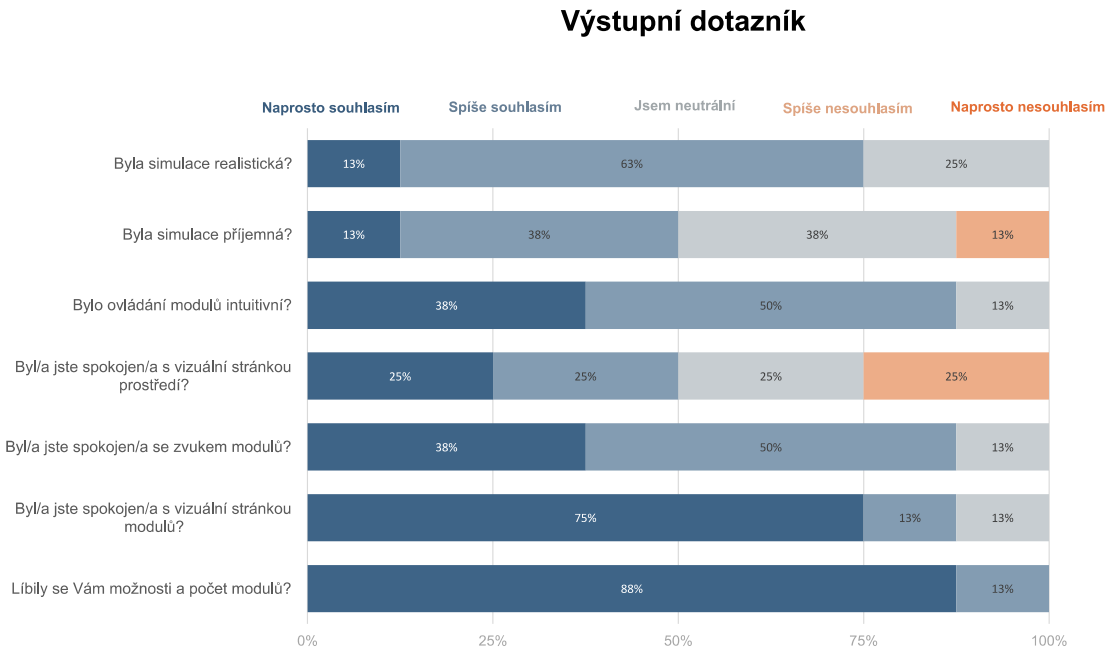
Z odpovědí můžeme vyčíst, že několik testujících spíše s virtuální realitou zkušených nebylo. To negativně ovlivnilo testování, jelikož nevolnosti z virtuální reality jsou výrazné obzvlášť u nezkušených jedinců [5].

Po prvním sezení jsem v reakci na připomínky k průběhu testování přidal porovnání scény tvořené dle návrhu a prázdné scény. Dále se ze začátku odkryly nedostatky v seznámení uživatelů s ovládáním. Jejich opravení vedlo k mnohem rychlejšímu splnění úkolu u dalších participantů. Celkově zpětná vazba vedla k přibližně čtyřiceti menším i větším změnám v aplikaci. Tyto změny obsahovaly přidání zpětné vazby k několika interakcím, změny vizuální podoby a omezení pohybu všech knoflíků, kompletní změna vysvětlivek, změna fyziky kabelů, upravení pozic objektů v pracovní stanici, změna několika stavových textů, změna Pure Data patche distortion modulu (k silnějšímu zkreslení), změna pozic generovaných objektů a menu, změna ovládání menu, změna přicvakávání kabelů k modulům a v neposlední řadě opravení několika nalezených chyb ve funkcionalitě.

Mezi zajímavá zjištění může patřit fakt, že všichni testující preferovali plynulé otáčení oproti skákavému a plynulý pohyb oproti teleportování (navzdory radám k minimalizaci nevolnosti v [5]). Dále testující z pohledu nevolnosti preferovali prázdnou scénu, ačkoliv neimplementovala žádné praktiky o správných scénách pro virtuální realitu. Méně objektů totiž vedlo k vyšší snímkové frekvenci. I přes to se jim líbila možnost výběru mezi oběma scénami (předposlední testující uvádí, že preferuje přírodní scénu ke kreativní činnosti a prázdnou k činnosti praktické).

Po seznámením se všemi druhy interakce byl zbytek testování (tedy ovládání modulů) velice svižný. Ale právě ono seznámení představovalo největší překážku. Do budoucna by bylo vhodné seznámení rozšířit do několika levelů, kde by se každý level zaměřoval na osvojení jedné interakce.

Na závěr přidávám výsledky výstupního dotazníku 5.3 a 5.4. Můžeme z ní vyčíst, že největší úspěch měl u participantů hlavní cíl této práce - moduly syntezátoru.



Obrázek 5.3. Výsledky Likert části výstupního dotazníku.

Otázky	Tester 1	Tester 2	Tester 3	Tester 4
Co se Vám nejvíce líbilo?	Chňapání kabelů	možnosti modulů	Velmi reálně prostředí a propracované detaily	Zvuková stránka modulů
Co se Vám nejméně líbilo?	Teleport	individualizace prostředí a	na pohyb, to pro mě nebylo zcela intuitivní	Rotace hlavy, spínačů
Otázky	**Tester 5**	**Tester 6**	**Tester 7**	**Tester 8**
Co se Vám nejvíce líbilo?	Možnosti hrát si s moduly ve VR	snapování modulů	Změna zvuku při otočení	Imerzivita
Co se Vám nejméně líbilo?	Klepání obrazu	ovládání - pohyb	Přepínání otáčení při vstupu do menu	Externě - headset

Obrázek 5.4. Výsledky otevřené části výstupního dotazníku.

Kapitola 6
Budoucnost

Snad nejzajímavější rozšíření mé práce by mohla být možnost vytvářet moduly přímo ve virtuální realitě. V nynější verzi může přidání jednoduchého modulu trvat zkušenému vývojáři přibližně jeden den včetně testování. Tento proces však obsahuje externí tvorbu Pure Data patche, tvorbu vizuální podoby modulu, tvorbu Blueprintové třídy, C++ třídy a kompilaci. LibPD, které nyní slouží jako médium mezi C++ kódem aplikace a Pure Daty, neumožňuje tvorbu patchů. Navrhuji tedy dvě možná řešení tohoto problému.

Nejprve představím svůj vlastní návrh tvorby Pure Data patche. Patche, které LibPD interpretuje, se ukládají v podobě jednoduchých čitelných textových souborů. Podívejme se na textovou podobu patche pro reproduktor.

```
#N canvas 0 0 1693 850 10;
#X obj 40 50 dac~;
#X obj 40 10 adc~;
#X connect 1 0 0 0;
#X connect 1 1 0 1;
```

Patch nejdříve definuje velikost plátna, na kterém jsou objekty položeny při grafickém programování patche. Dále na plátno na souřadnice x a y pokládá vyjmenované objekty. Nakonec příkazem connect tvoří propojení prvního výstupu prvního objektu do prvního vstupu druhého objektu a druhého výstupu prvního objektu do druhého vstupu druhého objektu. Unreal Engine nám dovoluje ukládat textové soubory v C++ části kódu. Bylo by možné vytvořit grafické objekty představující různé funkce Pure Dat a kabely pro jejich propojení. Dále systém, který by dovolil uživateli tyto objekty generovat a například na nějakém plátně ve hře spojovat. Systém by postupně četl objekty, které uživatel umístil, a tvořil by řetězec písmen s příhodnými příkazy obj a connect, které jsme viděli v patchi reproduktoru. Tento řetězec přeposílá do C++ a tam se ukládá do textového souboru s koncovkou .pd. Dále musíme vytvořit třídy, které patch využijí. Možný postup je vytvořit univerzální Actor třídu modulu. Obsahovala by připravené C++ třídy, čekající na jméno patch souboru. Tento Actor, by obsahoval několik vstupů, několik knoflíků a výstup. Tyto komponenty mají předem propojené odpovídající funkce v C++ a správně pojmenované posluchače pro Pure Data. Uživatel může za běhu aplikace nepotřebné komponenty pomocí funkce Component->UnregisterComponent() odebírat. Po odebrání komponentů nastaví uživatel jméno své třídě a ta finalizuje tvorbu inicializací LibPD. Návrh je limitovaný předem definovanými komponenty univerzálního modulu. Také si musíme uvědomit, že netvoříme novou třídu, ale pouze upravujeme instanci univerzálního modulu.

Druhou možností by mohlo být podrobné studium projektů Purrdata a Plugdata. Oba projekty implementují vlastní GUI pro Pure Data. Plugdata obsahují Heavy kompilátor zmiňovaný v kapitole 4.1.1, otevírají tedy možnost jiného propojení Unreal Engine tříd a Pure Dat oproti systému implementovaném v této práci.

Toto představené rozšíření by expandovalo cílovou skupinu o lidi zabývající se návrhem hardware Eurorack modulů.

Kapitola 7
Závěr

Zadání práce bylo vytvoření modulárního syntezátoru ve virtuální realitě s použitím Pure Data patchů pro zprostředkování zvuku.

Pomocí zabalení knihovny LibPD do pluginu pro Unreal Engine se tento cíl podařilo naplnit. Dále se podařilo naplnit funkční požadavky zadané v návrhu 3.6. Celkově aplikace obsahuje 11 objektů využívajících Pure Data a další dva moduly představující modulační zdroje pro ostatní moduly. Vytvořena byla i scéna odpovídající návrhu. Aplikace umožňuje ukládání a generování objektů z menu.

Průběh tvorby začínal v rámci Samostatného projektu analýzou a vytvořením grafické podoby modulů a scény. Ačkoliv byly tyto části zpětně měněny, vytvořily dobrý základ pro další směřování projektu. Vytvoření modulů a interakcí bylo nečekaně přímočaré (ačkoliv se při testování objevily možnosti pro zlepšení). Zdaleka nejnáročnější bylo však zprovoznění komunikace mezi softwarem Pure Data a Unreal Enginem, a to hned z několika důvodů. Nejdříve bylo náročné vybrat mezi možnostmi propojení, jelikož každá představovala nějaká omezení ve funkcionalitě a každá měla své vlastní výzvy. Před tvorbou této práce jsem neměl dostatečné zkušenosti v kompilaci jazyka C a C++, a z toho důvodu se zprovoznění vybraného LibPD a následná tvorba pluginu ukázala jako náročný problém. Unreal Engine navíc, z mého osobního pohledu, nemá zcela perfektní dokumentaci. Po prvním úspěšně poslaném zvuku bylo náročné navrhnout systém propojení modulů, ale se zkušenostmi z předchozího studia (obzvlášť algoritmizace) jsem byl schopen tento systém vytvořit.

Ačkoliv byly jednotlivé moduly inspirovány moduly od firmy Moog, nabízí několik nových funkcí. Všechny moduly operují s nejméně dvěma kanály, a tedy nabízí možnost pro stereofonie. Modulů si uživatel může vytvořit mnohem více, než s prostorově omezeným hardwarem. Jednoznačnou výhodou nad hardwarovými moduly je dostupnost a možnost jednoduchého rozšíření - v nynější podobě aplikace není po úvodním proškolení náročné přidávat do systému nové moduly.

Aplikace byla podrobena testování a dle získané zpětné vazby upravena. Z testování vyšlo několik překvapujících zjištění ohledně preferencí uživatelů. Nicméně testování potvrdilo dosažení cílů, jelikož participanti ocenili možnosti modulů a jejich interakce. Jako místo pro zlepšení se ukázal pohyb uživatele a právě scéna vytvořená dle návrhu. Scéna, která měla uživateli pomoct s orientací ve virtuálním prostoru, byla v některých případech brána jako zbytečná a odváděla pozornost od důležité části aplikace - od syntezátoru. Dále zhoršovala plynulost běhu aplikace. I přes toto úskalí participanti popisovali zájem o delší pobyt v aplikaci a další zkoumání zvukových možností.

Výsledek práce představuje inovativní implementaci elektronického hudebního nástroje ve virtuální realitě s možností experimentování se zvukem. Vytvořený most mezi enginem a softwarem Pure Data zpřístupňuje pro muzikanty a sound designery schopnosti Pure Dat v novém světě. V nynějším stavu je tento projekt praktickým nástrojem a další rozšíření ve směrech optimalizace a nových možností z něj může udělat skutečně jedinečnou technologii v svém oboru.

V současné podobě je tento projekt praktickým nástrojem. S pomocí optimalizace a dalších rozšíření má potenciál stát se skutečně jedinečnou technologií ve svém oboru.

Literatura

[1] Jason Gregory. *Game Engine Architecture*. A K Peters Ltd, 2015. http://ce.eng.usc.ac.ir/files/1511334027376.pdf.

[2] A. Andrade. *Game engines: a survey*. 2015. https://www.researchgate.net/publication/283657797_Game_engines_a_survey.

[3] Miller S. Puckette. *Pure Data: another integrated computer music environment*. 1997. https://msp.ucsd.edu/Publications/icmc97.pdf.

[4] Johannes Kreidler. *Programming Electronic Music in Pd*. 2009. http://www.pd-tutorial.com/english/index.html. [cit. 2024-04-15].

[5] Jason Jerald. *The VR Book: Human-Centered Design for Virtual Reality*. Edition 1 vyd. ACM Books, 2016. ISBN 978-1-97000-113-6.

[6] Intel. *Exploring the Intel Manufacturing Environment through Augmented Reality*. https://www.youtube.com/watch?v=GWsvKM8eVVc. [cit. 2024-14-5].

[7] David Crombie. *THE COMPLETE SYNTHESIZER*. Omnibus Pr Schirmer Trade Books, 1984. https://exellon.net/book/The_Complete_Synthesizer.pdf.

[8] Alexander Palumbo, Graham Zonta a Michael Wakefield. *Modular reality: Analogues of patching in immersive space*. 2020. https://www.tandfonline.com/doi/abs/10.1080/09298215.2019.1706583.

[9] Alexander Palumbo, Graham Zonta a Michael Wakefield. *Affordances and Constraints of Modular Synthesis in Virtual Reality*. 2020. https://www.nime.org/proceedings/2020/nime2020_paper106.pdf.

[10] Daniel Rothmann. *SynthVR LinkedIn popis*. https://www.linkedin.com/in/danielrothmann/?originalSubdomain=dk. [cit. 2023-12-28].

[11] Daniel Rothmann. *SynthVR Modules*. https://github.com/42tones/SynthVR-Modules/blob/master/README.md. [cit. 2023-12-28].

[12] 42tones. *SynthVR*. https://store.steampowered.com/app/1517890/Syn-thVR/. [cit. 2023-12-15].

[13] SteamDB. *SynthSpace SteamDB stránka*. https://steamdb.info/app/1355640/info/. [cit. 2023-12-28].

[14] Bright Light Interstellar. *SynthSpace SteamDB stránka*. https://github.com/brightlightrx/synthspace-audio-layer. [cit. 2023-12--28].

[15] Bright Light Interstellar Limited. *SYNTHSPACE*. https://store.steampowered.com/app/1355640/SYN-THSPACE/. [cit. 2023-12--15].

[16] Robin Vincent. *Synthmulator: a playable and patchable Eurorack modular in a virtual world*.
https://www.gearnews.com/synthmulator-a-playable-and-patchable-eurorack-modular-in-a-virtual-world/. [cit. 2023-12-28].

[17] Luka Osborne. *Play a Modular Synth in Virtual Reality With 'Synthmulator'*.
https://enmoreaudio.com/play--a-modular-synth-in-virtual-reality-with-synthmulator/. [cit. 2023-12-15].

[18] Hannes Zeller a Ludwig Barfuss. *OpenSoundLab – A virtual sound laboratory for the arts*. 2022.
https://dl.acm.org/doi/abs/10.1145/3561212.3561249.

[19] Bonseung Kim, Jaehwa Koo, Kwangsu Yoon a Nahye Cho. *Understanding the Formation of User's First Impression on an Interface Design from a Neurophysiological Perspective - EEG Pilot Study*.
https://dl.acm.org/doi/10.17210/hcik.2016.01.139.

[20] Joseph J. Jr. LaViola. *A Discussion of Cybersickness in Virtual Environments*.
https://www.eecs.ucf.edu/~jjl/pubs/cybersick.pdf.

[21] Ramotion. *VR in UX Design: Basic Guidelines for a Better Experience*.
https://www.ramotion.com/blog/vr-in-ux-design/. [cit. 2023-12-15].

[22] Blake Hudelson. *Designing for VR: A Beginners Guide*.
https://marvelapp.com/blog/designing-vr-beginners-guide/. [cit. 2023-12--15].

[23] Ajay Sawant a Vishal Rana. *Game Engine Market*. 2023.
https://www.aimarketreport.com/smart-technologies/game-engine-market. [cit. 2024-04-15].

[24] Liutaio Mottola. *Table of Musical Notes and Their Frequencies and Wavelengths*.
https://www.liutaiomottola.com/formulae/freqtab.htm. [cit. 2024-05-3].

[25] Bob Moog Foundation. *Zrození Moog Theremins*.
https://artsandculture.google.com/story/XQVBysUYsQr1GQ. [cit. 2024-04--20].

[26] Joshua Statzer. *The VR Expansion Plugin*.
https://vreue4.com/. [cit. 2024-05-3].

[27] Igor Dall'Avanzi. *Pure Data patches in Unreal Engine 4 via Wwise*.
https://igordallavanzi.wixsite.com/gamenoise/single-post/2017/03/23/pure-data-patches-in-unreal-engine-4-via-wwise. [cit. 2024-05-3].

[28] Wasted Audio. *Heavy Compiler Collection*.
https://github.com/Wasted-Audio/hvcc. [cit. 2024-05-3].

[29] Dan Brinkmann, Tal Wilcox, Richard Kirshboim, Ryan Eakin a Peter Alexander. *libpd: Past, Present, and Future of Embedding Pure Data*. 2016.
http://www.danomatika.com/publications/libpd_pdcon_16.pdf.

Příloha A
Vstupní dotazník

Stupnice:

1. Naprosto souhlasím
2. Spíše souhlasím
3. Jsem neutrální
4. Spíše nesouhlasím
5. Naprosto nesouhlasím

S použitím čísel ze stupnice odpovězte na následující otázky:

a) Jste zkušení v oboru modulární syntézy?
b) Jste zkušení v ovládání virtuální reality?
c) Máte rádi elektronické hudební nástroje?
d) Jste kreativní?

Na následující otázky odpovězte ano/ne:

e) Máte vlastní modulární syntezátor?
f) Máte vlastní brýle pro virtuální realitu?

Následují otevřené otázky:

h) Jak často používáte virtuální realitu?
ch) Jaký je váš oblíbený hudební žánr?
i) Pokud jste muzikant, na jaký hudební nástroj hrajete?

Příloha B
Výstupní dotazník

Stupnice:

1. Naprosto souhlasím
2. Spíše souhlasím
3. Jsem neutrální
4. Spíše nesouhlasím
5. Naprosto nesouhlasím

S použitím čísel ze stupnice odpovězte na následující otázky:

a) Líbily se Vám možnosti a počet modulů?
b) Byl/a jste spokojen/a s vizuální stránkou modulů?
c) Byl/a jste spokojen/a se zvukem modulů?
d) Byl/a jste spokojen/a s vizuální stránkou prostředí?
f) Bylo ovládání modulů intuitivní?
g) Byla simulace příjemná?
h) Byla simulace realistická?

Otevřené otázky:

ch) Co se Vám nejvíce líbilo?
i) Co se Vám nejméně líbilo?