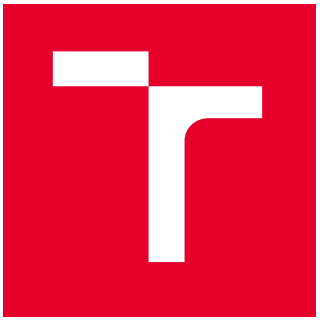
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2026 Tomáš Štrba



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

EXPERIMENTÁLNÍ SOFTWAROVÝ HUDEBNÍ NÁSTROJ PRO OPERAČNÍ SYSTÉM ANDROID S NEOBVYKLÝM GRAFICKÝM ROZHRANÍM

EXPERIMENTAL SOFTWARE MUSICAL INSTRUMENT FOR THE ANDROID OPERATING SYSTEM WITH AN UNUSUAL GRAPHICAL INTERFACE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Tomáš Štrba**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.**

SUPERVISOR

BRNO 2026

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Tomáš Štrba ***ID:*** 217703
Ročník: 3 ***Akademický rok:*** 2025/26

NÁZEV TÉMATU:

Experimentální softwarový hudební nástroj pro operační systém Android s neobvyklým grafickým rozhraním

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je realizovat plně funkční prototyp nástroje, navrženého v semestrální práci.

DOPORUČENÁ LITERATURA:

[1] HOLMES, Thom. Electronic and experimental music: technology, music, and culture. 3rd ed. New York: Routledge, 2008. ISBN 978-0-415-95782-3.
[2] PUCKETTE, Miller. The theory and technique of electronic music. Hackensack, NJ: World Scientific Publishing, c2007. ISBN 978-9812700773.

Termín zadání: 9.2.2026 ***Termín odevzdání:*** 2.6.2026

Vedoucí práce: doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Cieľom tejto práce je vytvorenie experimentálneho softvérového hudobného nástroja pre mobilnú platformu Android. Nezvyčajnosť tejto aplikácie spočíva v spojení jednej z najsúčasnejších metód tvorby zvuku, granulárnej syntézy, s interaktívnym používateľským rozhraním umožňujúcim modifikovať parametre granulárnej syntézy v reálnom čase na úrovni pripomínajúcej hru. Aplikácia využíva pre tvorbu a úpravu zvuku vizuálny programovací jazyk Pure Data pomocou platformy Plugdata a pre tvorbu interaktívneho užívateľského prostredia spolu s vizuálnou odozvou v reálnom čase herný engine Unity. Prvé kapitoly slúžia na stručné objasnenie operačného systému Android, programovacieho jazyka Pure Data, platformy Plugdata, Heavy Compliler Collection (hvcc) a herného enginu Unity. Ďalej je predstavený samotný návrh aplikácie a kompletná realizácia aplikácie.

KĽÚČOVÉ SLOVÁ

experimentálny hudobný nástroj, Unity, Plugdata, kompilátor hvcc, Android, granulárna syntéza

ABSTRACT

The aim of this thesis is to create an experimental software musical instrument for the Android mobile platform. The uniqueness of this application lies in the combination of one of the most contemporary methods of sound synthesis — granular synthesis — with an interactive user interface enabling real-time modification of granular synthesis parameters at a level reminiscent of a game. The application utilizes the visual programming language Pure Data via the PlugData platform for sound creation and processing, and the Unity game engine for the development of the interactive user interface along with real-time visual feedback. The first chapters provide a brief overview of the Android operating system, the Pure Data programming language, the PlugData platform, the Heavy Compiler Collection (hvcc), and the Unity game engine. Subsequently, the design of the application and its complete implementation are presented.

KEYWORDS

experimental musical instrument, Unity, Plugdata, hvcc compiler, Android, granular synthesis

Vysadené pomocou balíčku `thesis` verzie 4.13; `https://latex.fekt.vut.cz/`

ŠTRBA, Tomáš. *Experimentální softwarový hudební nástroj pro operační systém Android s neobvyklým grafickým rozhraním*. Bakalárska práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2026. Vedúci práce: doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Tomáš Štrba
VUT ID autora:	217703
Typ práce:	Bakalárska práca
Akademický rok:	2025/26
Téma záverečnej práce:	Experimentální softwarový hudební nástroj pro operační systém Android s neobvyklým grafickým rozhraním

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce. V súvislosti s vytvorením záverečnej práce som neporušil zásady a odporúčania VUT na využívanie generatívnej umelej inteligencie.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Rád by som poďakoval vedúcemu bakalárskej práce pánu doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D. za odborné vedenie, konzultácie a včasné odpovede k práci.

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

Cieľom tejto práce je návrh a nasledovná realizácia experimentálneho softvérového hudobného nástroja pre mobilnú platformu Android, ktorý spája jednu z najsúčasnejších metód tvorby zvuku, granulárnu syntézu, s interaktívnym používateľským rozhraním umožňujúcim modifikovať parametre granulárnej syntézy v reálnom čase na úrovni pripomínajúcej hru. Zároveň je aplikácia spracovaná na najdostupnejšie hardvérové zariadenie, mobilný telefón. Pre tvorbu a úpravu zvuku využíva vizuálny programovací jazyk Pure Data pomocou platformy Plugdata a pre tvorbu interaktívneho používateľského prostredia spolu s vizuálnou odozvou v reálnom čase herný engine Unity.

Používanie Pure Data pre tvorbu Android aplikácie nie je žiadna novinka, avšak kombinácia herného enginu Unity a Pure Data je v praxi už výrazne ojedinelé spojenie napriek tomu, že Unity poskytuje veľa možností pri tvorbe takéhoto typu aplikácie.

Nástroj je schopný generovať náhodné zvuky, ktoré sú reprezentované kruhmi na obrazovke a meniť určité parametre zvuku pohybom týchto kruhov, či už manuálne dotykom na kruhy alebo automaticky. Je taktiež navrhnutý tak, aby dokázal vloženie vlastnej zvukovej nahrávky a následovne interaktívne pracoval s granulárnou syntézou tejto zvukovej nahrávky pomocou ďalších objektov a tým menil zvukové parametre. Toto všetko je taktiež obohatené rôznymi zvukovými a vizuálnymi efektmi.

Aplikácie podobného typu už existujú, ale sú postavené na ovládaní, ktoré by človek bez znalostí nevedel využiť na plný potenciál. Nezvyčajnosť tejto aplikácie spočíva práve v kombinácii interaktívneho používania rôznych objektov na zmenu parametrov zvuku, ktoré generujú kruhy a granulárna syntéza. V kombinácii s vizuálnymi efektmi predstavuje táto aplikácia niečo ako medzikrok medzi hrou a plnohodnotným nástrojom na tvorbu hudby.

V prvých častiach práce je stručne objasnený operačný systém Android, programovací jazyk Pure Data, platforma Plugdata a Heavy Compliler Collection (hvcc), ktorý je potrebný na kompiláciu Pure Data patchu. Ďalej je predstavený herný engine Unity a vysvetlená granulárna syntéza. V následujúcich kapitolách je vysvetlený samotný návrh a realizácia aplikácie.

1 Operačný systém Android

Operačné systémy pre smartfóny mali mnoho rôznych implementácií a typov, ale ako je to pri väčšine technológií, len tie najvšestrannejšie prerazia do každodenného sveta. Operačný systém Android patrí medzi tie, ktoré sú najpoužívanejšie, spolu s iOS od firmy Apple Inc. a HarmonyOS od firmy Huawei. Tieto uvedené operačné systémy pre smartfóny tvoria hromadnú väčšinu používaných operačných systémov v dnešnej dobe, pričom operačný systém Android tvorí až 79% z celosvetového podielu predaja smartfónov [2]. Operačný systém Android je vyvíjaný firmou Google LLC, no samotný Android je open source platforma, čo pomáha k veľmi rýchlemu rozvoju.

1.1 Architektúra operačného systému Android

Architektúru operačného systému Android si je možné predstaviť, ako zloženie viacerých rôznych softvérov, ktoré spolu tvoria logické a vizuálne spojenie operačného systému. V operačnom systéme Android sa toto zloženie delí na päť vrstiev, a to: jadro operačného systému, knižnice, android Runtime, vývojová platforma a základné aplikácie [3].

Jadro operačného systému

Jadro operačného systému vychádza z Linuxu verzie 2.6 a pokrýva nasledujúce funkcie: riadenie procesov, správa pamäte, správa virtuálnej pamäte, správa napájania, sieťové pripojenie a správa zariadení [4]. Jeho ďalšou funkciu je priama komunikácia s hardvérovou častou smartfónu.

Knižnice

Ďalšou vrstvou sú knižnice systému Android, ktoré umožnia spracovávanie rôznych typov dát ako napríklad: SQLite, WebKit, Media framework, OpenGL. Tieto knižnice sú napísané v programovacom jazyku `C` alebo `C++`.

Android Runtime

Android Runtime je tretia z týchto piatich vrstiev a je zložená z virtuálneho stroja Dalvik a základných knižníc pre `Javu`. Dalvik umožňuje operačnému systému spracovávať jednotlivé aplikácie v Androide ako samostatné objekty virtuálneho stroja. Táto skutočnosť prispieva k celkovej stabilite, úsporu energie a bezpečnosti operačného systému Android.

Vývojová platforma

Vývojová platforma poskytuje služby, ktoré pomáhajú vývojárom pre Android zakomponovať pomocou rozhrania pre programovanie aplikácií ich aplikácie spolu so systémovými službami ako napríklad: *activity manager*, *content providers*, *telephony manager*, *location manager*, *resource manager* [4].

Základné aplikácie

Základné aplikácie je vrstva, s ktorou sa najčastejšie stretávame a predstavuje predinštalované aplikácie na smartfóne ako napríklad: kamera, list kontaktov, internetový vyhľadávač, alebo dodatočne nainštalované aplikácie rôzneho druhu.

1.2 Android SDK a NDK

Android SDK je široká zbierka nástrojov s hlavným cieľom vyvíjať aplikácie. Je určená primárne na prácu s kódom napísaním v `Jave` alebo `Kotline`. Pozostáva z viacerých častí, medzi ktoré patrí vývojová platforma, kompilátor, ladiaci program [5]. Keďže Android SDK je balík primárne určený na prácu s programovacími jazykmi ako je `Java` a `Kotlin`, ktoré disponujú obmedzeným prístupom k fyzickým komponentom zariadenia a iným hardvérovým funkciám [6], bol postupom času vyvinutý Android NDK.

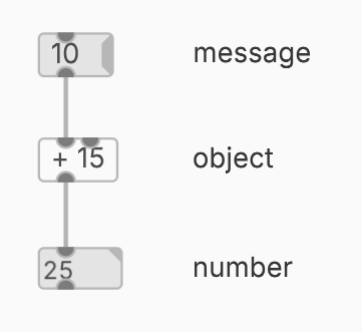
Android NDK umožňuje vývoj aplikácii pomocou programovacieho jazyku `C` alebo `C++`, ktoré sú následovne prostredníctvom Android NDK skompilované do použiteľného natívneho kódu a knižníc, ktoré sú následovne volané v rámci Java kódu pomocou Java Native Interface [7]. Existencia Android NDK zabezpečuje o niečo lepší výkon kódu, ale čo je o mnoho viac podstatnejšie je fakt, že je teraz možné použiť a implementovať obrovské množstvo kódu, ktorého by bez Android NDK nebolo možné využiť vo vývoji Android aplikácií. Toto je taktiež kľúčová skutočnosť pri realizácii spojenia Unity cez patch v prostredí PlugData, o ktorom bude nasledujúca kapitola.

2 Pure Data a Plugdata

2.1 Pure Data

Pred tým ako bude vysvetlené, čo je to Plugdata, tak je potrebné uviesť do popredia pojem Pure Data. Pure Data je vizuálny programovací jazyk vyvinutý Millerom Puckettom v 90. rokoch 20. storočia na prácu primárne pre tvorbu hudobne interaktívnych prác.

Práca v Pure Data je zostavená z používania troch základných stavebných blokov alebo boxov nazývaných *message*, *object* a *number*. Používateľ tieto jednotlivé boxy spája pomocou ich výstupov a vstup. Počet výstupov a vstupov je konkrétne definované pre každý jeden typ boxu [8]. Korektným spojením rôznych typov boxov vznikne takzvaný `.pd` patch, ktorý je uložený ako obyčajný textový súbor.



Obr. 2.1: Pure Data patch v Plugdata.

Ako je z obrázku 2.1 vidieť, tak tok údajov má smer zhora dole, kde kliknutím na message box sa pošle údaj do object boxu, ktorý má pridelenú triedu pre sčítanie znamienkom `+` a ako argument má číslo `15`. Pure data ďalej prevedie danú matematickú operáciu a do number boxu dosadí číslo `25`.

Veľmi dôležitou častou je taktiež rozdelenie object boxu na dva ďalšie typy. Typ, ktorý pracuje so zvukovým signálom a typ, ktorý nepracuje so zvukovým signálom, nazývaný *control object box*, uvedený v obrázku 2.1. Typ, ktorý pracuje so zvukovým signálom sa taktiež nazýva *signal object box* a vyznačuje sa použitím vlnovky za triedu použitú v object boxe ako napríklad `[osc~]`. Dátový tok má určité pravidlá a vo väčšine prípadoch nesmie prúdiť zo signálového object boxu naspäť do control object boxu. Existujú výnimky ako napríklad control object box `[ftom]`, ktorý konvertuje frekvenciu na MIDI, alebo control object box `[snapshot~]`, ktorý konvertuje signál na float číslo vždy, keď je spustený control object boxom `[bang]` [8].

2.2 Plugdata

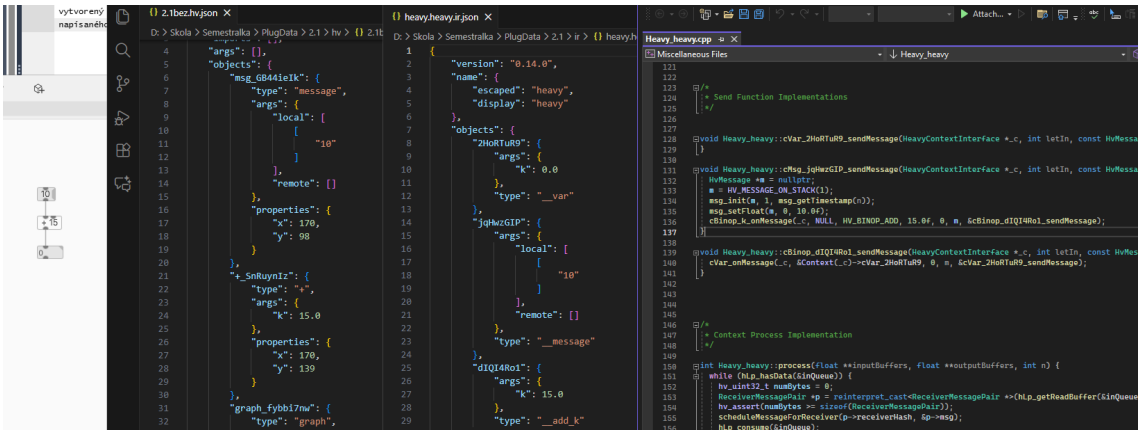
Plugdata je teda plugin wrapper od Timothy Schoen, ktorý začal byť vyvíjaný v novembri roku 2021 a získal veľkú pozornosť z rád užívateľov Pure Data, ktorí v dnešnej dobe taktiež prispievajú k jeho inovácii, keďže Plugdata je open source platforma [9]. Plugdata je teda novodobá interpretácia pre Pure Data od Millera Pucketta.

Hlavnou myšlienkou pre vývoj Plugdata bolo zmodernizovanie grafického rozhrania a poskytnutie vylepšeného prostredia, s ktorým dokáže užívateľ pracovať. Ďalšou výhodou je taktiež veľké množstvo predinštalovaných knižníc, ktoré pridávajú množstvo objektov ako napríklad knižnica `ELSE` a `cyclone`.

No najväčšou výhodou akú Plugdata v porovnaní s Pure Data prináša, je možnosť integrácie `.pd` patchov pomocou vstavanej kompilácie. Poskytuje možnosť z `.pd` patchov vytvoriť VST2, VST3 plugin alebo ho priamo zabudovať do hardvérového zariadenia a mnoho ďalšieho. Tento kompilátor sa nazýva Heavy Compiler Collection alebo skratkou hvcc.

2.2.1 Heavy Compiler Collection (hvcc)

Heavy Compiler Collection je kompilátor pôvodne napísaný v programovacom jazyku `Python` a vytvorený od Enzien Audio [10]. Jeho hlavnou funkciou je generácia programovacieho kódu napísaného v programovacom jazyku `C` a `C++` z `.pd` patchu, ktorý je možné ďalej aplikovať pomocou generátora špecifického pre rôzne softvéry a programovacie jazyky ako je napríklad: Daisy, DPF, FMOD, Javascript, OWL, Pdext, Wwise, Unity [10]. Pri prvotnom generovaní sa vytvoria tri typy súborov ako je vidieť na obrázku 2.2.



Obr. 2.2: Kompilácia z Plugdata do použiteľného kódu pre Unity.

Súbor `hv`, ktorý predstavuje reprezentáciu `.pd` patchu v zdrojovom programovacom jazyku `Heavy`, je prvým z týchto súborov. Pred tým ako môže byť `hv` súbor posunutý ďalej na spracovanie, je potrebné spraviť takzvaný medzikrok použitím súboru `ir`. IR (Intermediate Represantation), často preložené aj ako medzijazyk, slúži na optimalizáciu zdrojového kódu a následovné spracovanie do výsledného kódu v súbore `c`. Tu už je finálna verzia napísaná v požadovanom `C` a `C++` jazyku.

2.2.2 Heavylib

Potrebné je si taktiež uvedomiť, že tento prevod z `.pd` patchu do `C` a `C++` jazyku nie je bezchybný. Taktiež nevie podporovať všetky typy box objektov, či už to je objekt, ktorý spracováva audio signál alebo iba objekt, ktorý slúži na zmenu dát [11]. Preto sa taktiež pri vyvíjaní kompilátoru myslelo na vývoj knižnice `Heavylib` [12], ktorá obsahuje kompatibilné abstrakcie niektorých pôvodných Pure Data objektov a niekoľko kompletne nových.

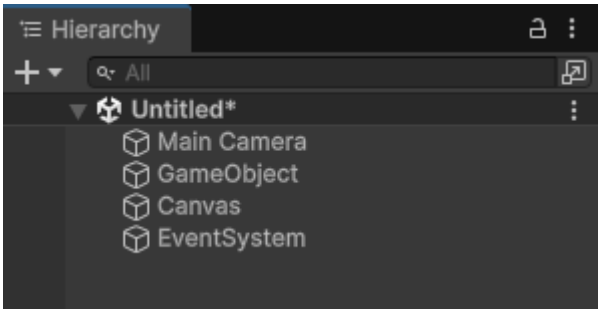
Pre syntézu sú to objekty oscilátorov, nízkofrekvenčných oscilátorov a obálky ako napríklad: `[hv.osc~ saw]`, `[hv.lfo sine]`, `[hv.pinknoise~]`, `[hv.vline~]`. Pre signálové spracovanie sú to objekty efektov, filtrov alebo dynamiky ako napríklad: `[hv.compressor~]`, `[hv.reverb~]`, `[hv.filter~ low/highpass]`. A nakoniec taktiež existujú rôzne objekty pre matematické operácie. Pretože knižnica `Heavylib` existuje, strata niektorých originálnych Pure Data objektov nie je ku koncu tak výrazne detrimentálna.

3 Unity

Unity je herný engine, ktorý bol vydaný v roku 2005 a je vyvíjaný spoločnosťou Unity Technologies [13]. Programovací jazyk použitý na vývoj Unity je `C` a `C#` je použitý ako skriptovací jazyk. Aktuálna verzia Unity je Unity 6 spustená v roku 2024. Budovanie hier v prostredí Unity je realizovateľné pre viaceré platformy, od stolného počítača a mobilného zariadenia až po virtuálnu realitu a webové prostredie. Unity je primárne používaný na vývoj 2D a 3D hier. Práca v Unity sa odohráva v Unity editore, ktorý je tvorený z viacerých okien alebo sekcií, kde medzi hlavné patrí hierarchia, scéna, hra, inšpektor, projekt a konzola.

Hierarchia

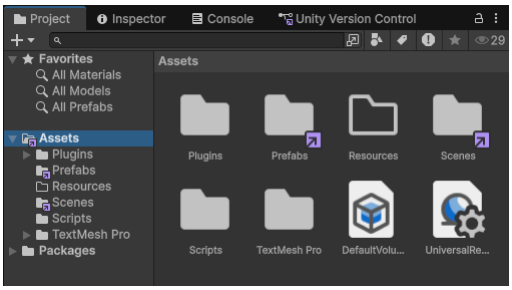
V sekcii hierarchia sa nachádzajú všetky herné objekty, ktoré sú práve obsiahnuté v aktuálnej scéne. Tieto objekty je možné zorganizovať do určitej hierarchie typu rodič-dieťa.



Obr. 3.1: Hierarchia

Projekt

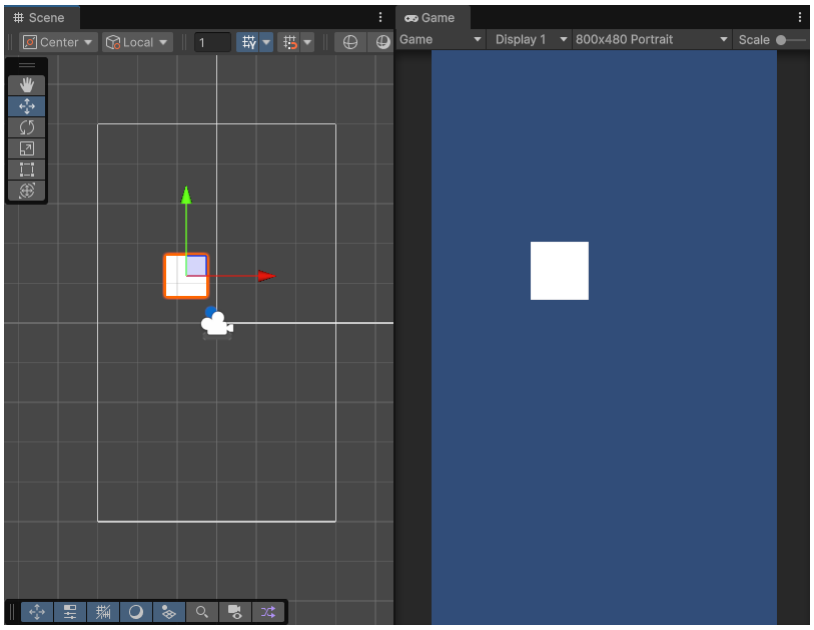
Obsahuje všetky súbory v projekte. Každý typ súboru má explicitne priradený priečinok, aby Unity vedel tieto súbory správne prideliť pri generovaní hry.



Obr. 3.2: Projekt

Scéna a Hra

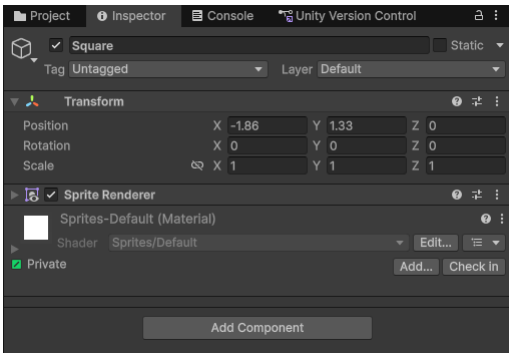
Scéna a hra sú prvky, ktoré idú ruka v ruke. Do scény používateľ vkladá herné objekty, ktoré bude mať hráč na svojej obrazovke. Vytvára a dizajnuje grafické prostredie, dáva mu formu a tvar. Zatiaľ čo v hre môže simulovať celkové správanie hry z pohľadu hráča, testovať audio alebo logiku rôznych interakcií.



Obr. 3.3: Scéna

Inšpektor

Inšpektor sa stará o nastavenia jednotlivých herných objektov a taktiež umožňuje pridávanie rôznych komponentov ako je napríklad: animácia, audio, efekty, skripty alebo transform ako je vidieť na obrázku 3.4.



Obr. 3.4: Inšpektor

4 Granulárna syntéza

K tvorbe zvuku na základe syntézy existuje viacero rôznych prístupov, ktoré ľudia používajú už desiatky rokov na vytvorenie umelého zvuku. Riadením, skladaním a upravovaním signálov je možné reprodukovať každý jeden zvuk. Na rozdiel od aditívnej syntézy, ktorá vychádza zo sčítania jednoduchých signálov výhradne sínusového typu, a subtraktívnej syntézy, ktorá je založená na použití spektrálne bohatšieho signálu ako je napríklad biely šum alebo iného širokospektrálneho signálu, z ktorého sa následnou filtráciou vytvára nový signál, je granulárna syntéza koncipovaná na základe práce s už existujúcim zvukovým záznamom, s ktorým ďalej pracuje.

Granulárna syntéza je typ syntézy, ktorá pracuje s takzvanými zrnami. Zrná sú krátke zvukové úseky, ktoré majú dĺžku spravidla 1 až 100 ms. Práca s týmito krátkymi zrnami umožňuje zmeniť pôvodný signál a takmer úplne vytvoriť nový zvuk, nového charakteru. Termín takmer je použitý preto, lebo výsledný signál sa stále odráža od originálu, takže jeho spektrálne vlastnosti sú v určitých častiach podobné. Každé jedno zrno má vlastnú obálku, ktorá definuje čas nábehu a čas poklesu zrna aby sa odstránili nechcené signálové artefakty. Granulárna syntéza začne byť zaujímavá s pridávaním ďalších desiatok alebo stoviek zŕn, ktorým môžeme upravovať rôzne parametre ako je napríklad: hustota, dĺžka, rýchlosť generovania, rozsah rozmiestnenia, prekrývanie, výška tónu a pozícia.

Hustota nám určuje počet zŕn generovaných v rámci časovej jednotky. Prostredníctvom dĺžky je možné upraviť časový rozsah každého zrna. Ako už bolo spomínané, tak táto dĺžka je spravidla 1 až 100 ms. Rýchlosťou generovania definujeme frekvenciu vzniku zŕn v danom intervale. Rozsah rozmiestnenia určuje z akej časti zvukového záznamu sa budú generovať zrná. Prekrývanie špecifikuje rozsah časového prekryvu jednotlivých zŕn. Výška tónu jednoducho určuje ako vysoko alebo hlboko dané zrná znejú a pozíciou vieme udávať z akej časti sa začnú zrná generovať.

Uvedené parametre možno manuálne meniť, alebo generovať s určitom náhodnosťou. Práve spojenie týchto všetkých skutočností, je pomocou granulárnej syntézy možné dosiahnuť neslýchané zvuky s jedinečnými vlastnosťami. [14]

5 Návrh aplikácie

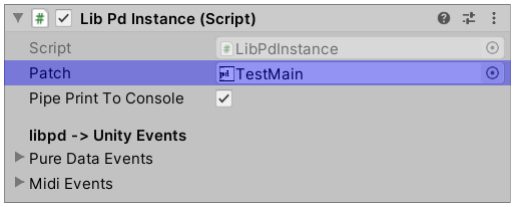
Ako bolo v úvode naznačené, tak navrhnutá aplikácia je schopná generovať náhodné zvuky reprezentované kruhmi a nasledovným pohybom týchto kruhov meniť určité parametre zvuku. Taktiež dokáže použiť vloženú nahrávku pri práci s granulárnou syntézou a pomocou objektov reprezentujúcich zrno granulárnej syntézy meniť parametre tejto syntézy. Toto všetko je taktiež obohatené rôznymi zvukovými a vizuálnymi efektmi.

Návrh aplikácie v rámci tejto bakalárskej práce prebiehal v prostredí Unity a Plugdata. Plugdata pôsobí ako nástroj, v ktorom sa vytvárala syntéza od základov pre použitie v Unity a Unity slúžil ako hlavná platforma pre vytvorenie interaktívneho užívateľského prostredia. V tejto kapitole budú opísané jednotlivé kroky podniknuté pri návrhu aplikácie, prvotné užívateľské rozhranie, časti kódu napísaného v Unity pomocou `C#` a ukážky Plugdata `.pd` patchu.

5.1 Výber softvéru

Ako už bolo na začiatku tejto kapitoly naznačené, tak vývoj aplikácie prebiehal v prostredí Unity a Plugdata, no cesta k výberu práve týchto dvoch softvérov nebola vôbec priama.

Prvotné návrhy sa odrážali od existencie projektov a hier ako FRACT OSC [15], ktoré už pred viac ako jedenástimi rokmi dokázali zakomponovať Pure Data do Unity pomocou knižnice `LibPd Integration` [1]. Spomínaná knižnica dokáže priamo používať `.pd` patch vytvorený v Pure Data a priradiť ho do skript komponentu v Unity ako je vidieť na obrázku 5.1.



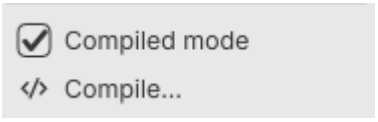
Obr. 5.1: `LibPd Integration` skript komponent v Unity [1].

Prvotné návrhy sa teda konštruovali cez práve zmienenú knižnicu `LibPd`, ktorá fungovala a podarilo sa vniesť `.pd` patch do Unity a pracovať s ním v Unity editore. Spomínaná knižnica nemala žiadne viditeľné nedostatky a podporovala všetky potrebné Pure Data objekty. Po otestovaní v Unity editore nasledovalo samotné stavanie android aplikácie, kde `LibPd Intergration` knižnica zlyhala a muselo sa

prejsť k inej variante, ktorou je použitie viackrát spomínaného kompilátoru hvcc zabudovaného v prostredí Plugdata.

5.2 Plugdata

V rámci Plugdata sú riešené základné zvuky, ktoré môže používateľ ovládať dotykom prsta na kruhové objekty. Pri tvorení `pd.` patchov je potrebné brať na vedomie, že existuje skupina nepodporovaných objektov [11], ktoré hvcc nedokáže skompilovať. Výhodou tvorenia `.pd` patchov v Plugdata je však prítomnosť takzvaného kompilačného režimu.

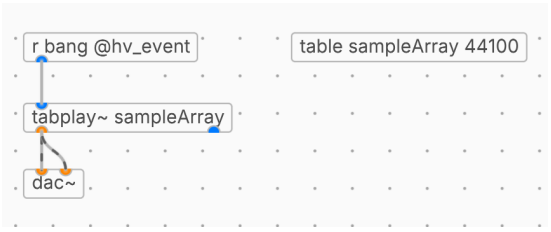


Obr. 5.2: Kompilačný režim v Plugdata.

Spomínaný kompilačný režim zaručí, že všetky použité objekty budú podporované pri kompilácii pomocou hvcc. Kompilačný režim sa jednoducho zapína v menu Plugdata, ako je vidieť na obrázku 5.2.

5.2.1 Integrácia Plugdata do Unity

Integrácia Plugdata do Unity je zaručená pomocou takzvaných sprístupnených parametrov a udalostí. V originále sa nazývajú *exposed events* a *exposed parameters*. V aplikácii sú takto vytvorené sprístupnené udalosti potom ovládané v Unity pomocou skriptov, ako je vidieť na obrázku 5.3.



(a) Bang event a audioklip.

```
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class SampleLoader : MonoBehaviour
{
    public Hv_heavy_AudioLib heavy;
    public AudioClip clip;
    public string tableName = "sampleArray";

    Unity Message | 0 references
    void Start()
    {
        heavy = GetComponent<Hv_heavy_AudioLib>();

        heavy.FillTableWithMonoAudioClip(tableName, clip);
        heavy.SendEvent(Hv_heavy_AudioLib.Event.Bang);
    }
}
```

(b) Bang event a audioklip.

Obr. 5.3: Plugdata Unity

Na obrázku 5.3a je vidieť vytvorený objekt bang event `[r bang @hv_event]`, ktorý je možné zavolať priamo v Unity skripte. Na obrázku 5.3b je v skripte aplikácie použitý nasledovný kód `heavy.SendEvent(Hv_heavy_Audiolob.Event.Bang)`. Takýmto spôsobom sa taktiež vytvorí v komponente nášho skriptu bang tlačítko, ktoré je možné manuálne stlačiť pri testovaní zvuku.

Popri bang eventu je taktiež realizovateľné načítanie zvukovej stopy priamo z Unity do Plugdata príkazom z hvcc knižnice. V aplikácii je to riešené pomocou zavolania príkazu `heavy.FillTableWithMonoAudioClip(tableName, clip)`. Týmto príkazom sa nám priradí audioklip `clip` do objektu `[table sampleArray 44100]`. Pomocou `tableName` si taktiež vieme konkrétne určiť o aký objekt sa presne jedná.

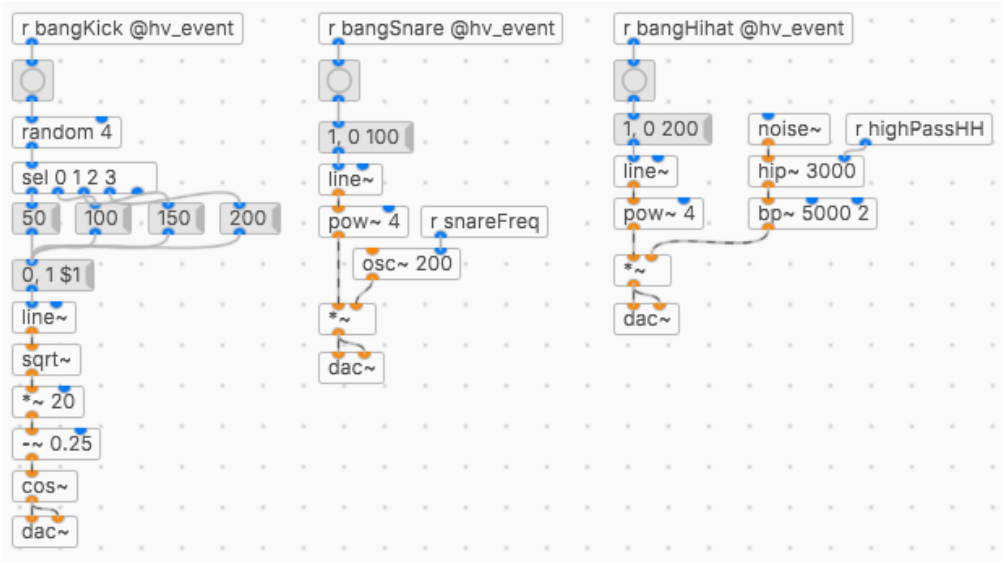
Medzi ďalšie sprístupnené parametre patria všetky send a receive objekty. Ako bolo spomenutú pri bang eventu, tak aj send a receive objektom je možné priradenie grafického prostredia v Unity komponente, s ktorým sa vie pri vývoji aplikácie pracovať. Slúži na to priradenie `@hv_param` v objekte recevie alebo send ako je napríklad `[r freq @hv_param 0 1 0.5]`. Takýmto priradením `@hv_param` vznikne v Unity komponente posuvný ovládač so základnou hodnotou 0,5 a rozsahom hodnôt od 0 po 1 s názvom `freq`.

Pri používaní týchto funkcií je taktiež vždy potrebné priradenie hvcc skriptu a následovné načítanie pomocou `public Hv_heavy_Audiolib heavy` a `GetComponent <Hv_heavy_Audiolib heavy>`. Zároveň je potrebné dodržiavať zhodnú vzorkovaciu frekvenciu medzi Unity a Plugdata, čo je vyriešené odoslaním vzorkovacej frekvencie a dĺžky nahrávky v samploch z Unity do `.pd` patchu.

5.2.2 Zvuky pre kruh

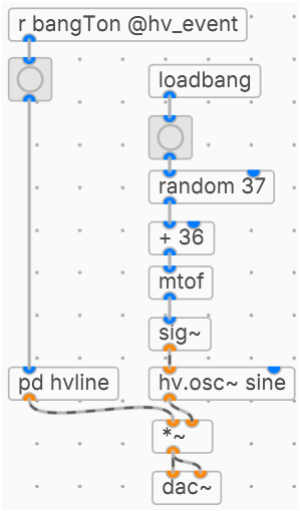
Zbierka základných zvukov, s ktorými používateľ pracuje sú navrhnuté pomocou jednoduchých `.pd` patchov. Keďže používateľ je schopný pridávať neurčité množstvo kruhov, s ktorými je možné pracovať, pridávať efekty a upravovať ich zvukové parametre, tak je možné vytvoriť pomerne plnú zvukovú nahrávku. V návrhu aplikácie sa pracuje s neurčitým množstvom kruhov, ale pri realizácii vzniknú bezpochyby limity vo výpočtovej technike, ktoré pri spracovaní veľkého počtu `.pd` patchov budú mať za následky nechcené signálové artefakty.

Na obrázku 5.4 je návrh perkusných zvukov typu kick, snare a hihat. Pri vytvorení kruhu sa priradí jeden z týchto troch zvukov na komponent zvukového zdroja v Unity. Pre zvuk typu kick je použitý objekt `[random 4]` a `[sel 0 1 2 3]`, ktorým sa náhodne vyberie čas nábehu zo štyroch rôznych možnosti. Ďalej sa vytvorí obálka, ktorá začne na 0 a postupne vzrastá na hodnotu 1 za práve spomínaný čas nábehu. Objekt `[line~]` premení číselnú správu na signál. Ďalej objekt `[sqrt~]` pomôže zjemniť nábeh a použitím `[cos~]` sa obálka transformuje na použiteľný signál.



Obr. 5.4: Návrh jednoduchých perkusných zvukov v Plugdata.

Podobne sú vytvorené aj zvuky typu snare a hihat. Pri vytváraní hihat bol použitý objekt `[noise~]`, ktorý produkuje biely šum. Následnou hornopriepustnou filtráciou s frekvenciou 3000Hz, ktorú je taktiež možné ovládať pomocou receive objektu a objektu pásmovej priepusti bolo docielené vytvorenie zvuku znejúceho ako hihat.



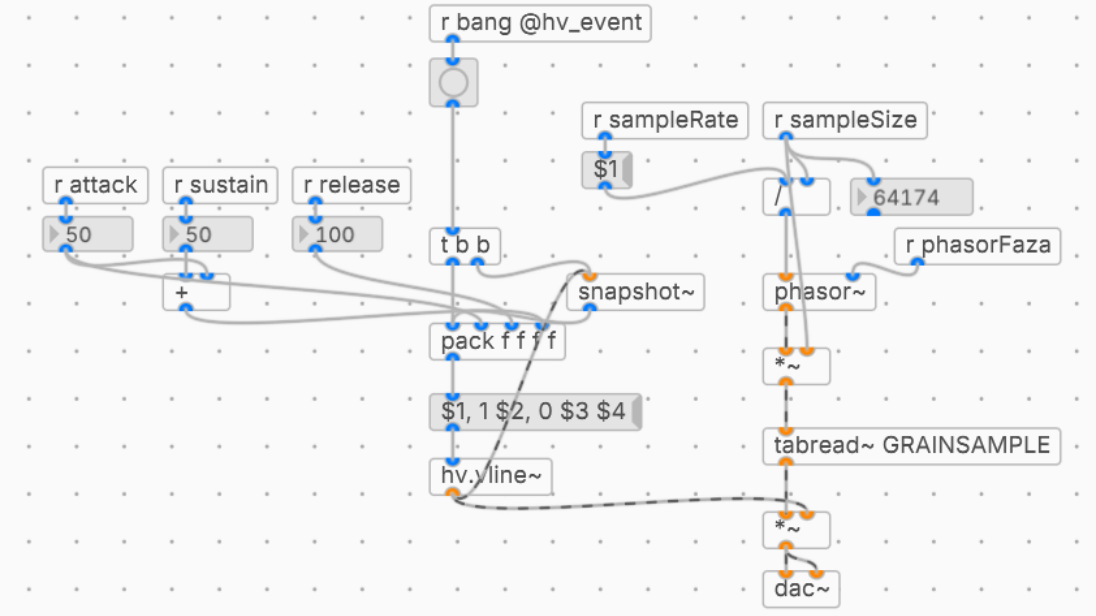
Obr. 5.5: Návrh jednoduchého tónu v Plugdata.

Na obrázku 5.5 je návrh jednoduchého tónu, ktorý po prvotnom načítaní na kruh vyberie náhodnú MIDI klávesu z rozsahu od 36 po 72, čo odpovedá notám od C2 po C5. Tento tón sa opätovne spúšťa pomocou bang eventu, ktorý je pripojený na objekt `[pd hvline]`. Objekt `[pd hvline]` obsahuje podobnú obálku ako je znázornená v obrázku 5.6.

5.2.3 Granulárna syntéza

Pri navrhovaní granulárnej syntézy je prvým krokom načítanie zvukového záznamu a následne určiť jeho vzorkovaciu frekvenciu a dĺžku v samploch. Hneď pri tomto kroku dochádza k spomínanej skutočnosti, že hvcc kompilátor nepodporuje všetky objekty a použitie typického objektu `[soundfiler]`, ktorého výstup je vzorkovacia frekvencia a dĺžka v samploch, nie je možné použiť. Metóda, ktorá bola zvolená je teda načítanie vzorkovacej frekvencie a dĺžky v samploch v Unity a následovne preposlať tieto informácie do `.pd` patchu.

Ďalším krokom každej granulárnej syntézy je generovanie zrna zo zvukového záznamu. Čítanie zo zvukového záznamu je spracované pomocou objektov `[phasor~]` a `[tabread~]`, ktoré po vypočítaní podielu vzorkovacieho kmitočtu a dĺžky v samploch je možné použiť na čítanie zvukového záznamu. Ďalej je treba vytvoriť ADSR obálku pre zrno a zasielaním bang eventu na rôzne hodnoty fáze do objektu `[phasor~]` získame prvé, veľmi základne zrno.



Obr. 5.6: Návrh zrna v Plugdata.

Na obrázku 5.6 je vidieť prvotný `.pd` patch návrh pre spracovanie základného zrna pre granulárnu syntézu zo zvukového záznamu. Obsahuje ako už spomínaný `[phasor~]`, ktorý cyklicky cez celý zvukový záznam frekvenciou danou pomocou receive objektov `[r sampleRate]` a `[r sampleSize]` prechádza. So všetkými receive objektmi je možné komunikovať cez Unity skript a tým meniť ich parametre ako je používateľovi vhodné.

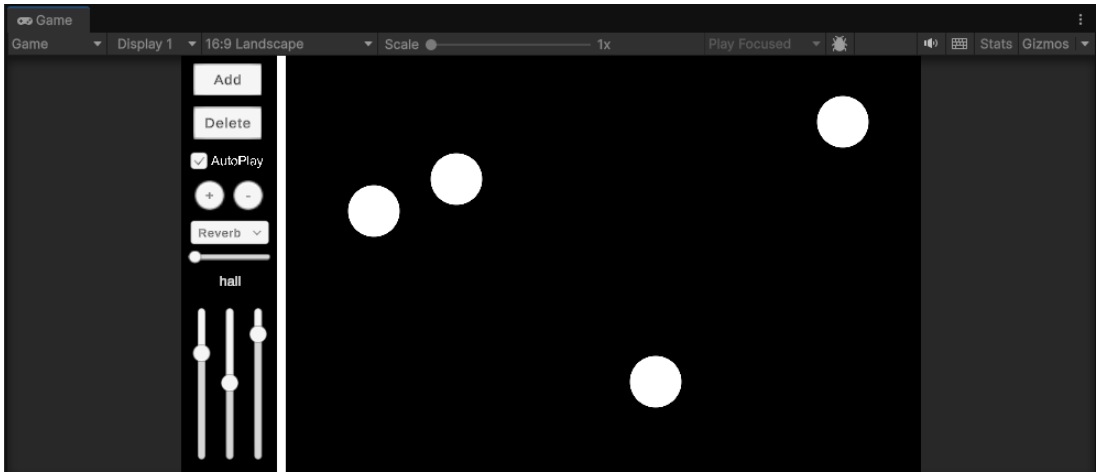
Medzi ďalšie parametre granulárnej syntézy, ktoré môže používateľ ovládať buď manuálne alebo automaticky patrí:

- hustota,
- dĺžka,
- rýchlosť generovania,
- výška tónu,
- pozícia,
- rozsah rozmiestnenia.

Pridanie ďalších zŕn je realizovateľné dvomi spôsobmi. Jeden z nich je priamo pomocou skopírovania návrhu z obrázku 5.6 v samotnom `.pd` patchy a tým mať v Unity jeden zvukový zdroj. Druhou možnosťou je priradenie zvukového zdroja do každého zrna, ktoré bude používateľ v aplikácii vidieť. Výber medzi týmito možnosťami bude závisieť od výpočtovej náročnosti na celú aplikáciu a preto sa v návrhu nedá presne určiť, ktorá metóda je vhodnejšia na použitie.

5.3 Unity

Interaktívne užívateľské prostredie, v ktorom je možné pridávanie, odstraňovanie, klikanie, posúvanie kruhov a dynamická úprava parametrov je kompletne navrhnutá cez herný engine Unity. V tejto sekcii sú znázornené prvotné návrhy kódu a základne ukázané používateľské grafické prostredie. Na obrázku 5.7 je zobrazený prvotný návrh aplikácie v Unity editore.



Obr. 5.7: Grafické rozhranie v Unity editore.

5.3.1 Pridávanie a odstraňovanie kruhov

K prvým objektom, s ktorými používateľ pracuje, patria prvky používateľského prostredia určené na pridávanie a odstraňovanie kruhov a tým aj pridávanie prvotného zvuku do aplikácie, ktorý je možný pomocou rôznej pozície kruhov manipulovať.

Výpis 5.1: Pridávanie kruhov.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.InputSystem.UI;
using System.Collections.Generic;

public class addcircleui : MonoBehaviour
{
    [Header("Circle Prefab")]
    public GameObject circle;

    public static List<GameObject> listofCircles =
    new List<GameObject>();

    public void AddCircle()
    {
        GameObject fabCircle = Instantiate(circle);
        listofCircles.Add(fabCircle);

        fabCircle.transform.position = new Vector2
        (Random.Range(-18f, 26f), Random.Range(-13f, 13f));
    }
}
```

Pridávanie kruhov je možné pomocou kliknutia na tlačidlo, ku ktorému je priradený skript komponent `addcircleui.cs`. Tento skript je vidieť z výpisu5.1. Jeho hlavnou úlohou je vytvorenie prefab (predpripraveného) objektu pomocou `GameObject fabCircle = Instantiate(circle)` na náhodnú pozíciu. Ako druhú úlohu vykonáva vytvorenie a následovné pridanie kruhov do určitého listu objektov, ktorý je použitý pri odstraňovaní kruhov ako je vidieť z výpisu 5.2.

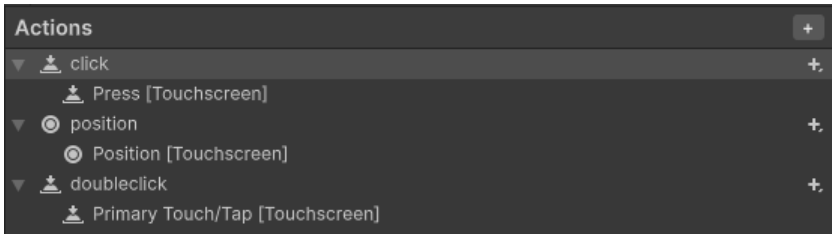
Výpis 5.2: Odstránenie kruhov.

```
public class removeCircleUi : MonoBehaviour
{
    public void RemoveCircle()
    {
        if (addcircleui.listofCircles.Count == 0) return;

        int lastIndex = addcircleui.listofCircles.Count - 1;
        GameObject last = addcircleui.listofCircles[lastIndex];
        Destroy(last);
        addcircleui.listofCircles.RemoveAt(lastIndex);
    }
}
```

Verejná trieda `removeCircleUI` sa nám teda stará o odstránenie posledného kruhu z listu objektov a taktiež o samotné odstránenie interaktívneho kruhu z našej obrazovky a tým aj zvuku tvoreného takýmto kruhom.

5.3.2 Dotyk kruhu

Kruh je naprogramovaný tak, že vie spracovať a rozoznávať tri rôzne typy dotykov. Tento užívateľský vstup je spracovaný pomocou Unity InputActionEditoru ako je vidieť na obrázku 5.8.



Obr. 5.8: InputActionEditor

Prvým je jeden krátky dotyk, ktorý vyšle bang event do .pd patchu a tým prehrá zvuk v ňom uložený a zároveň zmení na 250 ms farbu kruhu ako odozvu pre užívateľa. Druhým je dotyk a držanie, čím je možné kruh posúvať po obrazovke v časti aplikácie vyhradenej pre pohyb objektov. A tretím je rýchly dvojitý dotyk. Pokiaľ dvojitý dotyk na kruh trvá menej ako 250 ms, tak sa zapne automatické posielanie bang eventov a tým aj prehrávanie zvuku. Rýchlosť posielania bang eventov a tým aj celková rýchlosť prehrávania zvuku je ovplyvnená pozíciou kruhu na obrazovke.

5.3.3 Vplyv pozície kruhu na zvuk

Pred spustením automatického prehrávania je aplikáciu, presnejšie kruhy, zvuk z kruhov a rôzne parametre zvuku, možné ovládať pomocou pozície kruhu na obrazovke. Kruh rozoznáva svoju polohu v kartézskej sústave XY a podľa nej upravuje parametre zvuku. Medzi tieto paramentre patrí napríklad: výška zvuku, rýchlosť posielania bang eventov, množstvo dozvuku a oneskorenia. V každom momente je možné upravovať iba dva rôzne typy parametrov. Výber parametru je možné pomocou menu.

Výpis 5.3: Výška zvuku.

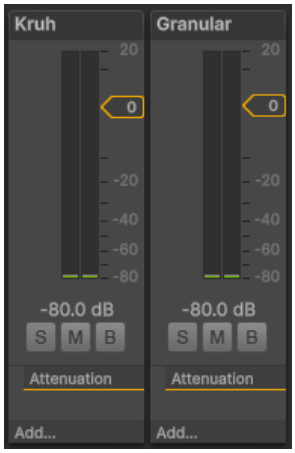
```
float yReshaped = Mathf.InverseLerp(maxY, minY, y);                    1
audioSource.pitch = Mathf.Lerp(pitchmaxY, pitchminY, yReshaped);       2
```

Z výpisu 5.3 je vidieť kód, ktorý má na starosť výšku zvuku. Výška zvuku sa mení iba v závislosti od Y-ovej súradnice kruhu, čo je vidieť priradením `float y` na transform pozíciu Y-ovej súradnice. Pomocou `Mathf.InverseLerp` a `Mathf.Lerp` sa vypočíta pozícia medzi 0 a 1 a následovne medzi určenou hodnotou `pitchmaxY` a `pitchminY`. Podobným spôsobom sú riešené aj ostatné parametre.

Ďalšou možnosťou je taktiež ovládanie rôznych zvukových parametrov, ktoré boli vytvorené pomocou `.pd` patchu a obsahujú na svojom vstupe receive objekt ako napríklad `[r frequency]`. Do takto použitého receive objektu je teda dovolený zápis zo skriptu v Unity, ktorý je možné upravovať v reálnom čase.

5.3.4 Mixážny pult

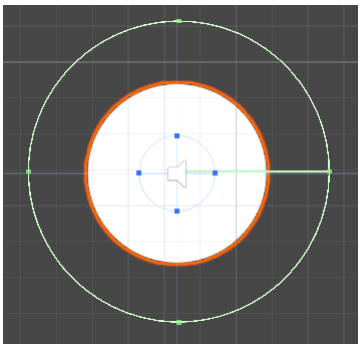
Ovládanie hlasitosti zvuku je navrhnuté pomocou posuvného ovládača, prostredníctvom ktorého si môže používateľ prispôsobiť hlasitosť zvuku kruhu, alebo samotnej granulárnej syntézy. Mixážny pult je vytvorený pomocou zabudovaného mixážneho pultu, ktorý je možné špecificky priradiť ku každému komponentu zvukového zdroja, čo je v tomto prípade každý jeden kruh a zrno granulárnej syntézy. Keďže v Unity úroveň hlasitosti mixážneho pultu spadá od -80 dB do 20 dB a obyčajný posuvný ovládač pracuje s hodnotami od 0 po 1, tak je potreba transformovať tieto hodnoty a zároveň použiť logaritmickú mierku, aby mohol užívateľ správne pracovať s týmto prvkom. Na obrázku 5.9 je vidieť, ako vyzerá mixážny pult v Unity editore. Tento mixážny pult obsahuje taktiež tlačidlá *solo* a *mute*, ktoré možno do aplikácie taktiež pridať.



Obr. 5.9: Mixážny pult v Unity.

5.3.5 Automatické prehrávanie

Automatické prehrávanie sa spustí pomocou prepínača k tomu určenému, ktorý uvedie kruhy do pohybu. K ich pohybu dochádza z dvoch dôvodov. Prvý spočíva v tom, že súčasťou každého kruhu je komponent *RigidBody2D*, ktorý objektu poskytuje fyzikálne vlastnosti. Druhý je priradenie náhodnej sily a náhodného smeru na objekt. Takto sa začnú kruhy náhodne pohybovať a budú vznikať kolízie medzi objektami a medzi objektami a stenami.



Obr. 5.10: *Collider2D* okolo kruhu.

Medzi ďalší Unity komponent, ktorý nám pomôže pri automatickom prehrávaní je *Collider2D*. Na obrázku 5.10 je reprezentovaný zeleným kruhom. Takto pri zrážke dvoch kruhov alebo oblastí s ďalším collider komponentom je možné s touto udalosťou ďalej pracovať a v prípade tejto aplikácie slúži na poslanie bang eventu na spustenie zvuku. Ovládanie rýchlosti pohybu kruhov je možné pomocou posuvného ovládača.

5.3.6 Granulárna syntéza

Granulárna syntéza je v aplikácii zobrazená ako malý kruh, okolo ktorého je prstenec. Oba objekty majú okolo seba collider komponent, ktorým sa môže opäť posielať bang event, alebo určovať dobu, kedy sa budú obmieňať určité parametre zrna. Do prázdnej oblasti medzi vnútorným kruhom a prstencom sa budú generovať stlačením určeného tlačítka jednotlivé zrná granulárnej syntézy.

5.3.7 Zvukové efekty

Na výsledný zvuk z kruhov alebo z granulárnej syntézy je signál možné rozšíriť o rôzne zvukové efekty. Tieto efekty sú implementované pomocou základných zvukových efektov, ktoré Unity poskytuje. Medzi tieto efekty partia:

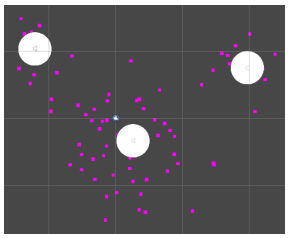
- lowpass (dolnopriepustný filter),

- highpass (hornopriepustný filter),
- delay (oneskorenie),
- flanger,
- distortion (skreslenie),
- pitch shifter (zmena tónovej výšky),
- chorus,
- compressor (kompresor).

Tieto efekty si môže vybrať užívateľ z menu, ktoré sa nachádza pod posuvným ovládačom pre úroveň hlasitosti. Väčšina týchto efektov obsahuje päť až desať parametrov, ktoré možno v Unity ovládať. To by znamenalo, že pri integrácii týchto parametrom by muselo byť vytvorené veľké množstvo posuvných ovládačov, čo by odporovalo prvotnému návrhu, podľa ktorého má aplikácia obsahovať čo najmenej prvok používateľského rozhrania. Takže je tento problém vyriešený pomocou jediného posuvného ovládača, ktorým je po voľbe určitého efektu možné prechádzať medzi prednastavenými nastaveniami.

5.3.8 Vizuálne efekty

Keďže Unity je primárne engine na vytváranie hier, tak je prirodzené, že obsahuje veľmi veľké množstvo vizuálnych efektov, ktoré je možné využiť na poskytnutie spätnej väzby používateľovi. V aplikácii to je viditeľné vďaka zmene farby kruhov pri kolízii dvoch objektov s collider komponentom alebo pri kliknutí na kruh. Prítomné je taktiež použitie robustného časticového systému, ktoré Unity poskytuje. Častice je možné upraviť do takej miery, že svojimi vlastnosťami dokážu do určitej miery reprezentovať samotné zvuky, ktoré sú generované. To je dosiahnuteľné napríklad odoslaním údajov z obálky ADSR a použitím týchto údajov na nastavenie parametrov častíc. V návrhu je použitý parameter attack ako hodnota, ktorá určuje rýchlosť pohybu častíc. Parameter decay ako hodnota, ktorá určuje dĺžku vzdialenia od kruhu. Parameter sustain ako hodnota, ktorá určuje životnosť častíc a parameter release ako hodnota, ktorá určuje jas častíc. Na obrázku 5.11 je vidieť veľmi základné použitie častíc v Unity editore.



Obr. 5.11: Častice okolo kruhu.

6 Realizácia aplikácie

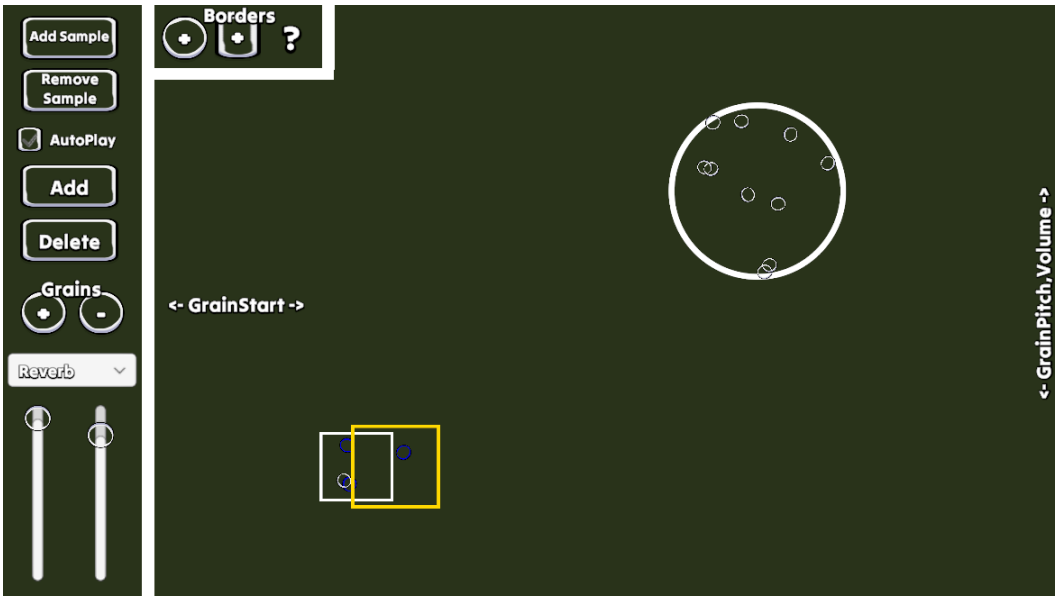
Pri realizácii aplikácie sa postupovalo podľa pripraveného návrhu spracovaného počas semestrálnej práce. Návrh pri realizácii poskytol dostatočne veľa základných informácii o integrácii Plugdata do Unity, s ktorými sa dalo ďalej pracovať. Avšak po spracovaní potrebných súborov pomocou hvcc a následnou kompiláciou, s ktorými nebol problém, nastala jedna z prvých prekážok pri realizácii. A to skutočnosť, že pri testovaní ľubovoľne zložitých či jednoduchých `.pd` patchov, muselo prebehnúť práve spomínané spracovanie súborov odznova. Táto komplikácia pridala pri testovaní rôznych `.pd` patchov dodatočnú prácu navyše, pre ktorú sa napokon rozhodlo spracovať použité `.pd` patche v aplikácii čo najjednoduchším spôsobom. To znamená, že použité `.pd` patche pracujú výhradne s hodnotami, nevyhnutnými ku bezchybnému chodu aplikácie a všetky logické alebo matematické výpočty sú kompletne naprogramované pomocou `C#` v prostredí Unity.

Jednou z najvýznamnejších zmien realizovaných v priebehu implementácie, je sústredenie sa na prácu s experimentálnou granulárnou syntézou, pričom práca s kruhmi zostala ako sekundárne zameranie aplikácie s funkciami z pôvodného návrhu.

6.1 Používateľské grafické rozhranie

Primárnym zámerom na použitie herného enginu Unity, bolo vytvorenie interaktívneho grafického rozhrania, ktoré by pomohlo s tvorivosťou pri práci s touto aplikáciou. Myšlienka spočívala v tom, aby aplikácia obsahovala čo najmenej nastavení a tvárila sa, ako keby bola hrou. Návrh z obrázku 5.7 sa blízko zhoduje s finálnou podobou aplikácie.

Na obrázku 6.1 je vidieť finálnu podobu používateľského grafického rozhrania. Skladá sa z tlačidiel a posuvných prvkov. Prvky *Add Sample* a *Remove Sample* slúžia na otvorenie prehliadača súborov v danom zariadení, spustení príslušných metód a odstránení uloženého zvukového záznamu, ohraničenia a zŕn. Prvky *Add* a *Delete* vykonávajú pridávanie a odstraňovanie kruhov. Prvky *Grains* (zrná) + a - priraďujú zrnám uložený zvukový záznam a taktiež majú za úlohu pridávanie a odstraňovanie zŕn. Ďalej môžeme vidieť menu na výber efektov, v ktorom je práve na obrázku Reverb (dozvuk). Tieto efekty sa upravujú pomocou posuvného prvku, ktorý je vidieť ako druhý zľava. Prvý posuvný prvok slúži na kontrolu celkovej hlasitosti zvuku aplikácie a je vždy funkčný. Ako posledné sú tri tlačidlá na pridávanie rôznych typov ohraničenia.



Obr. 6.1: Grafické rozhranie.

6.2 Granulárna syntéza

Hlavnou myšlienkou na použitie herného enginu Unity, bol práve návrh na spracovanie interaktívneho používateľského prostredia, ktorý nám dovolí pracovať s granulárnou syntézou nezvyčajným spôsobom. Pri samotnej realizácii sa vytvorený návrh v kapitole 5.3.6 prejavil ako správna cesta, ktorou je možné vizuálne riešiť tento problém a z praktického spracovania bol návrh v kapitole 5.2.3 dobrým štartovným bodom. Napriek tomu, sa pri realizácii objavilo podstatné množstvo malých aj veľkých problémov, ktoré sa nie vždy javili ako riešiteľné. Značné množstvo praktických riešení predstavených v kapitolách 5.3.1, 5.3.2 a 5.3.3 bolo taktiež použitých pri riešení granulárnej syntézy.

Z vizuálneho hľadiska je granulárna syntéza, tak ako bola opísaná v návrhu, zobrazená ako malý kruh, ktorý reprezentuje zrno granulárnej syntézy, okolo ktorého je prstenec alebo iný geometrický tvar, ako je napríklad štvorec. Tieto objekty medzi sebou komunikujú a pri zrážkach dávajú používateľovi spätnú väzbu vo forme spustenia zvukového záznamu a zmenou farby z bielej na modrú. Ak používateľ pracuje bez spustenia ovládacieho prvku autoplay, tak je v takom prípade možné posúvať prstenec. Posúvaný prstenec zmení farbu z bielej na červenú. Takouto formou môže používateľ prstenec posúvať po hracej ploche a udávať do pohybu jednotlivé zrná granulárnej syntézy. Prehrávaný zvukový záznam sa mení s polohou zŕn na hracej ploche, kedy os x ovplyvňuje začiatok a koniec prehrávania zo zvukového záznamu a os y ovplyvňuje hlasitosť a výšku zvukového záznamu.

6.2.1 Načítanie zvukového záznamu

Z praktického hľadiska je nutné začať pri granulárnej syntéze s načítaním zvukového záznamu. Načítanie zvukového záznamu je dosiahnuté pomocou Native File Picker pluginu od yasirkula [16] a taktiež pomocou UnityEngine.Networking knižnice. Predtým, ako je vôbec možné použiť tento plugin a knižnicu na spracovanie zvukového záznamu si musíme overiť, či máme prístup a práva spracovávať súbory z Android zariadenia. Ak nemáme, tak je treba tento prístup a práva vyžiadať ešte pred začiatkom načítavania zvukového záznamu. Ďalším krokom už teda je použitie Native File Picker pluginu a otvorenie prehliadača súborov v danom zariadení a následné spustenie korutiny s názvom `LoadAndPlayWav`.

Výpis 6.1: Použitie Native File Picker pluginu a korutiny.

```
public void OnPickWavButton()
{
    NativeFilePicker.PickFile(OnFilePicked, new string[] {
    "audio/wav", "audio/x-wav", "audio/wave" });
}

private void OnFilePicked(string path)
{
    StartCoroutine(LoadAndPlayWav(path));
}
```

`LoadAndPlayWav` pomocou `.GetAudioClip` z Unity knižnice Networking dokáže načítať potrebný zvukový záznam, ktorý sa potom ďalej spracuje do verejnej statickej premennej `GrainClip = clip`. Takýmto spôsobom môžu aj ostatné skripty komunikovať s touto premennou. Tieto jednotlivé metódy sa volajú, keď používateľ stlačí tlačidlo *Add Sample*.

Výpis 6.2: Korutina LoadAndPlayWav.

```
using (UnityWebRequest www = UnityWebRequestMultimedia.GetAudioClip
(url, AudioType.WAV))
{
    yield return www.SendWebRequest();

    if (www.result == UnityWebRequest.Result.Success)
    {
    AudioClip clip = DownloadHandlerAudioClip.GetContent(www);
    audioSource.clip = clip;
    GrainClip = clip;
    }
}
```

V ďalšom kroku už nastáva pridelenie spomínanej premennej `GrainClip` do zvukového bufferu `GRAINSAMPLE`, ktorý bol vytvorený v rámci príslušného `.pd` patchu. Pri tvorbe návrhu bol tento krok podrobne opísaný v kapitole 5.2.1. U prvotnej realizácii sa postupovalo rovnakou cestou, kedy sa použila metóda `heavy.FillTableWith MonoAudioClip` na pridelenie zvukového záznamu do bufferu. Tento postup funguje výhradne pre mono zvukové záznamy a výsledok takéhoto postupu pri použití stereo nahrávky by nebol v prípade tejto aplikácie pozitívny, pretože by vznikali rôzne deformácie zvukového záznamu, ako napríklad načítanie iba jednej polovice zvukového záznamu, spomaleniu či urýchleniu prehrávania. Druhá možnosť, ktorú je možné využiť, je použitie metódy `heavy.FillTableBuffer` ku ktorej už jednoducho nemôžeme použiť iba premennú `GrainClip`, ale musíme zvukový záznam previesť do poľa hodnôt ako je vidieť vo výpise 6.3. Toto nám zaručí správne fungovanie mono aj stereo zvukového záznamu.

Výpis 6.3: Metóda LoadClip.

```
public void LoadClip()
{
    AudioClip clip = WavFilePicker.GrainClip;
    if (clip == null) return;

    int monoSamples = clip.samples;
    float phasorFreq = (float)clip.frequency / monoSamples;

    float[] allData = new float[clip.samples * clip.channels];
    clip.GetData(allData, 0);

    float[] monoData = new float[monoSamples];
    for (int i = 0; i < monoSamples; i++)
    {
        if (clip.channels == 1)
            monoData[i] = allData[i];
        else
            monoData[i] = (allData[i * clip.channels] +
            allData[i * clip.channels + 1]) * 0.5f;
    }

    wrapper.FillTableBuffer("GRAINSAMPLE", monoData);
    wrapper.SendFloat("SampleSize", monoSamples);
    wrapper.SendFloat("PhasorFreq", phasorFreq);
}
```

Metóda `LoadClip` taktiež posiela pomocou `wrapper.SendFloat` ďalšie dve potrebné informácie ako je `SampleSize` a `PhasorFreq`.

6.2.2 Zrno granulárnej syntézy

Po správnom načítaní zvukového záznamu môže ´používateľ začať pracovať so samostatnou granulárnou syntézou. Zrná si môže používateľ, potom čo vytvorí ohraničenie, pridávať do hracej plochy pomocou tlačidiel *Grains* plus a mínus. Pridaním sa vygeneruje inštancia z prihotoveného objektu prefab pre zrno. Tento prefab obsahuje dokopy deväť komponentov. Medzi tieto komponenty patria:

- Transform,
- Spirit Renderer,
- Audio Source,
- Rigidbody 2D,
- Circle Collider 2D,
- Grain Autoplay, Sample, Wrapper a Heavy Audio skript.

Komponent *Transform* pracuje s pozíciou granule, s rotačnými vlastnosťami a s veľkosťou v danej scéne. Komponent *Spirit Renderer* zahŕňa vizuálne prvky. Komponent *Audio Source* obsahuje základné prvky na ovládanie zvuku a je nevyhnutný pre správne pracovanie *Heavy Audio* skriptu, taktiež nám umožní priradiť jednotlivé objekty so zvukovým zdrojom do mixážneho pultu, kde môžeme kontrolovať celkovú hlasitosť a pridávať rôzne efekty. Komponent *Rigidbody 2D* priradí fyzikálne vlastnosti a komponent *Circle Collider 2D* ohraničí oblasť, na základe ktorej môže Unity rozpoznávať kolíziu s inými objektmi.

`Grain Autoplay`, `Sample`, `Wrapper` a `Heavy Audio` predstavujú hlavné skripty, ktoré zabezpečujú správny chod aplikácie. Skript `Sample` bol už v podstate celkovo predstavený v predchádzajúcej podkapitole, pretože obsahuje iba jednu verejnú metódu s názvom `LoadClip` 6.3, ktorá vykonáva správne nahranie zvukového záznamu do `Heavy Audio` skriptu. `Wrapper` je jeden z pomocných skriptov, ktorého hlavnou úlohou je sprostredkovať komunikácia s metódami z `Heavy Audio` skriptov bez ohľadu nato, o ktorý presný heavy skript sa v určitom momente jedná. Existencia tohoto skriptu zjednodušila prístup k testovaní rôznych `.pd` patchov a zmenšila robustnosť ostatných skriptov. `Heavy Audio` skript je vytvorený pri prevode z `.pd` patchu pomocou hvcc ako bolo predstavené v kapitole 2.2.1.

Hlavná funkcia `Grain Autoplay` skriptu je zabezpečenie posielania bang eventu. Bang event sa posiela pri každej kolízii s iným objektom, ktorý má relevantný komponent *Collider2D*. V praxi to znamená, že sa bang event odošle pri kolízii s objektom ohraničenia a s ostatnými zrnami.

Hlavná časť zdrojového kódu `Grain Autoplay` skriptu je uvedená vo výpise 6.4. Na začiatku každej kolízii sa testuje pomocou metódy `CooldownBang`, či od poslednej kolízie prebehlo minimálne 50 ms. Ak áno, tak sa ďalej pokračuje, ak nie, bang event a ani žiadne iné zmeny nie sú vykonané. Toto je jeden zo spôsobov ako sa v aplikácii

snažíme zabrániť preťaženiu a následnému nechcenému zlyhaniu aplikácie. Keďže sa tento test koná pre každé jedno zrno zvlášť, tak nemá v praxi skoro žiadny dopad na hudobný výsledok aplikácie.

Medzi ďalšie funkcie vykonávané pri kolízii, je sledovanie pozície na ose x pomocou `rb.position.x`, ktorú následne zmeníme na hodnotu od 0 po 1 pomocou `Mathf.Clamp01(Mathf.InverseLerp(-21.07f, 30.26f, x))`. Túto hodnotu potom posielame prostredníctvom `wrapper.SendFloat` do heavy skriptu. Metóda `OnCollisionEnter2D` taktiež sleduje pozíciu na ose y. Hodnotu tejto pozície, tak ako u pozície x, spracuje na hodnotu od 0 po 1 a následovne upraví pre ovládanie hlasitosti a výšky zvuku. Nakoniec kód spustí korutínu `FlashColor`, ktorá pri kolízii zmení farbu zrna na modrú a naspäť na bielu.

Výpis 6.4: Metóda kolízie.

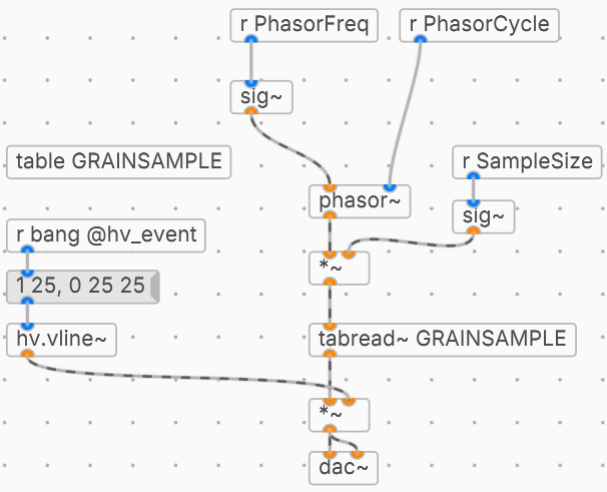
```
void OnCollisionEnter2D(Collision2D col)
{
    if (!CooldownBang()) return;

    rb.linearVelocity = Vector2.ClampMagnitude(rb.linearVelocity,
    moveSpeed);

    if (!AutoplayToggle.isPlaying)
    {
        float y = rb.position.y;
        float t = Mathf.InverseLerp(minY, maxY, y);

        AudioSource.volume = Mathf.Lerp(minVolume, maxVolume, t);
        AudioSource.pitch = Mathf.Lerp(minPitch, maxPitch, t);

        float x = rb.position.x;
        float grainPhase = Mathf.Clamp01
        (Mathf.InverseLerp(-21.07f, 30.26f, x));
        wrapper.SendFloat("PhasorCycle", grainPhase);
    }

    wrapper.SendBang();

    StartCoroutine(FlashColor());
}
```

6.2.3 Plugdata patch zrna

Návrh zrna v Plugdata na obrázku 5.6 sa oproti výslednej verzii 6.2 odlišuje pomerne málo. Celkový patch bol výrazne sprehľadnený a zjednodušený od návrhu. Väčšina

spojení bola prevedená do signálovej vrstvy a to aj pomocou objektu `[sig~]`, ktorý prevedie čísla do zvukového signálu. Tieto zmeny boli spravené primárne z dôvodu, že hvcc je prispôsobený hlavne na prácu s objektmi zo signálovej vrstvy a pri operáciach s číslami môže dochádzať k nežiadúcim javom.

Najväčšou zmenou oproti návrhu bola výsledná verzia ADSR obálky, na ktorú je pripojený hlavný bang event. ADSR obálka stratila možnosť dynamického zápisu hodnôt z dôvodu nestability systému, pričom sa prestúpilo k manuálnemu nastaveniu hodnôt s nábehom 25 ms a s dozvukom 25 ms. Táto zmena obmedzila jednu z možností, ktorá by vedela v pomerne veľkom množstve upraviť zvukový výstup aplikácie.

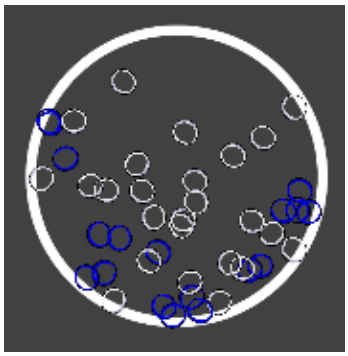


Obr. 6.2: Plugdata patch zrna.

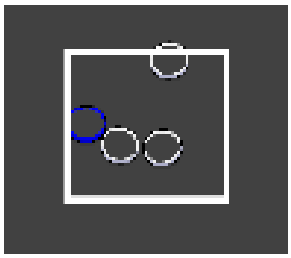
6.2.4 Práca s granulárnou syntézou

Potom čo si používateľ vyberie svoju zvukovú nahrávku, typ ohraničenia a vloží zrná, ako je napríklad zobrazené na obrázku 6.3a a 6.3b, môže začať pracovať a ovládať s týmito objektmi.

Medzi najjednoduchšiu formu ovládania takéhoto objektu patrí stlačenie a následné posunutie objektu po hracej ploche. Používateľ v skutočnosti dokáže posúvať iba ohraničenie, ktoré ale keď identifikuje kolíziu, posunie samostatné zrno ako bolo vysvetlené v kapitole 6.2.2. Posunutím objektu v ose x sa pri kolíziách posiela do heavy skriptu hodnota `PhasorCycle`, ktorý bol znázornený v patchy na obrázku 6.2 ako objekt `[r PhasorCycle]`. Týmto spôsobom sa upravuje pozícia začiatku prehrávania zvukovej nahrávky. K ďalším spôsobom ovládania patrí použitie dvoch prstov k zväčšeniu a k zmenšeniu ohraničenia. Zväčšenie a zmenšenie ohraničenia prináša do výsledného zvukového výstupu možnosť upravovať hustotu prehrávania



(a) Kruhové ohraničenie.



(b) Štvorcové ohraničenie.

Obr. 6.3: Granulárna syntéza.

zvukových objektov, a to bez nutnosti pridávania ďalších. Nadväzujúca alternatíva ako upravovať hustotu je taktiež možnosť pridania prázdneho ohraničenia a posunutia tohoto ohraničenia do priestoru iného. Týmto spôsobom obmedzíme priestor, v ktorom sa objekty pohybujú.

Keďže je každé jeden zvukový prvok priradený do mixážneho pultu, používateľ má možnosť používať zvukového efekty ako je napríklad: reverb, distortion, pitch, flanger, highpass a lowpass filter. Používateľ si vyberie jednotlivé efekty z menu a nastavením pravého posuvného prvku mení charakter efektu. Metóda `OnEffectSlider` 6.5 je priradené na posuvný prvok, ktorá volá ďalšiu metódu s názvom `ApplySlider`. Na výber nie sú v skutočnosti iba efekty z mixážneho pultu, ale aj napríklad premenná `moveSpeed`, ktorá špecifikuje hodnotu, ktorou sa všetky zrná v práve označenom ohraničení pohybujú.

Výpis 6.5: Metóda priradená na posuvný prvok.

```
public void OnEffectSlider(float value)
{
    ApplySlider(currentEffect, value);
}
private void ApplySlider(int index, float value)
{
    switch (index)
    {
        case 0: // Reverb
            Mixer.SetFloat(GranularReverbRoom,
            Mathf.Lerp(-10000f, 0f, Mathf.Pow(value, 0.2f)));
            break;
        case 1: // Distortion
            Mixer.SetFloat(Distortion, Mathf.Lerp(0f, 1f, value));
            break;
```

```
        case 2: // Pitch                                                    16
            Mixer.SetFloat(Pitch, Mathf.Lerp(0.2f, 1.6f, value));          17
            break;                                                         18
        case 3: // highpass                                                19
            Mixer.SetFloat(HPass, Mathf.Lerp(10f, 22000f, value));         20
            break;                                                         21
        case 4: // lowpass                                                 22
            Mixer.SetFloat(LPass, Mathf.Lerp(10f, 22000f, value));         23
            break;                                                         24
        case 5: // Flange                                                  25
            Mixer.SetFloat(Flange, Mathf.Lerp(0f, 20f, value));            26
            break;                                                         27
        case 6: // moveSpeed                                               28
            if (BorderManager.ActiveBorder == null) break;                 29
            float speed = Mathf.Lerp(1f, 30f, value);                      30
            foreach (GameObject grain in                                   31
            BorderManager.ActiveBorder.OwnedGrains)                        32
            {                                                              33
            if (grain == null) continue;                                   34
            GrainAutoplay autoplay = grain.GetComponent                    35
            <GrainAutoplay>();                                             36
            if (autoplay != null)                                          37
            autoplay.moveSpeed = speed;                                    38
            }                                                              39
            break;                                                         40
    }                                                                      41
}                                                                          42
```

6.3 Hudobné možnosti

Hlavná myšlienka tejto aplikácie je evokovať používateľovi ideu o tom, že sa nachádza v hracom prostredí a tým podporovať používateľovu kreativitu. To je docielené jednoduchým interaktívnym grafickým rozhraním, ktoré dáva používateľovi pri práci vizuálnu aj zvukovú odozvu.

Charakteristika výsledného zvuku závisí z veľkej časti od nahranej zvukovej nahrávky používateľa, s ktorou následovne pracuje v aplikácii. Vytvára granulárnu syntézy, ktorej prehrávanie je ovplyvnené fyzikálnymi vlastnosťami objektov herného enginu Unity. To znamená, že prehrávanie je do určitej miery generované náhodne a do určitej miery ovládané používateľom.

Prvý krok pri práci s aplikáciou je vybranie ohraničenia a následné pridanie zvukovej nahrávky. Už tento úvodný krok môže mať veľký vplyv na charakteristiku vyprodukovaných zvukov. Výber kruhového ohraničenia znamená, že začiatok

prehrávania jednotlivých prvkov nebude nikdy v súlade. Naopak výberom štvorcového ohraničenia môžeme docieliť, že začiatok prehrávania nektorých zŕn bude rovnaké. Tieto ohraničenia vieme posúvať, zmenšovať, zväčšovať alebo pridávať ďalšie pre prácu s viacerými nahrávkami naraz.

Aplikácia umožňuje používateľovi pracovať s viac ako jednou nahrávkou a to tým spôsobom, že stlačí jedno z ohraničení a pridá ďalšiu zvukovú nahrávku pomocou tlačidla *Add Sample*. To znamená, že je dosiahnuteľné granuly v jednej nahrávke upraviť tak, že sa pohybujú pomerne pomaly a v druhej nahrávke nastaviť na rýchly pohyb. Takáto jednoduchá úprava dokáže drasticky zmeniť výslednú charakteristiku prehrávania zvuku.

Zaujímavou možnosťou práce s aplikáciou je stlačenie prvku `AutoPlay`, ktorá uvedie objekty do pohybu a nastaví, aby sa zrná prehrávali cez celú zvukovú nahrávku. To pri vložení nenáročnej zvukovej nahrávky znamená, že je možné rozpoznať celý prehrávaný zdroj pomocou jednej granule s väčšou rýchlosťou, ale čo je zaujímavejšie je fakt, že každý prvok s heavy skriptom má osobitnú logiku, ktorá sa inicializuje pri pridaní. Takže pridaním ďalších zŕn, napríklad v strede zvukovej nahrávky, sa tieto nové objekty a teda aj výsledný zvuk začne miešať do seba a vytvárať vzájomne sa prekrývajúce zvukové vrstvy.

Všeobecne, fyzikálny prístup tejto aplikácie k spúšťaniu zvukov predstavuje jedinečný prvok, ktorý sa pomerne odlišuje od zaužívaných granulárnych syntetizérov, ktoré obyčajne pracujú a spúšťajú granuly prostredníctvom časových údajov alebo manuálnym vstupom používateľa. Takýto prístup je reprodukovateľný, ale v prípade tejto aplikácie je to pravý opak, ktorý je práve charakteristickým znakom experimentálnych nástrojov.

6.4 Možnosti rozšírenia

Medzi kľúčové oblasti ďalšieho rozšírenia aplikácie patrí predovšetkým funkcia nahrávania výsledného zvukového výstupu. To by nám dovolilo rýchlo experimentovať a následne použiť nahrávky v produkcii. Ďalšia z možností rozšírenia je vytvorenie pamäte na uloženie nastavení a pozície objektov pre budúce použitie.

Z hľadiska grafického rozhrania by bolo žiadúce rozšíriť podporu aplikácie pre rôzne typy rozlíšení pre zariadenia, ktoré používajú operačný systém Android, keďže v aktuálnej verzii môže vzniknúť pri niektorých zariadeniach kompletné obmedzenie používania aplikácie, v dôsledku nesprávnej pozície niektorých prvkov mimo viditeľnú plochu.

Stabilita aplikácie pri pracovaní s početným množstvom zŕn predstavuje ďalšiu z oblastí, ktorá by si zaslúžila zvýšenú pozornosť aj napriek tomu, že bolo podniknutých niekoľko krokov k zlepšeniu celkovej stability práve pri práci s vyšším počtom

zvukových objektov.

Podstatné rozšírenie, ktoré by výrazne obohatilo vyprodukovanú charakteristiku zvuku, je implementácia dynamickej ADSR obálky pre všetky zrná, ktoré sa nachádzajú v jednom ohraničení. Napriek tomu, že výsledná aplikácia má byť experimentálneho typu, tak by dobré obohatiť ovládanie zŕn aj o viac kontrolovateľnejší pohyb, ktorý by umožnil cieľavedomejšie vytváranie výsledného zvukového výstupu.

Záver

Cieľom tejto práce bol návrh a nasledovná realizácia experimentálneho softvérového hudobného nástroja pre mobilnú platformu Android, ktorý spája granulárnu syntézu s interaktívnym používateľským rozhraním umožňujúcim modifikovať parametre v reálnom čase na úrovni pripomínajúcej hru. Nezvyčajnosť tejto aplikácie spočíva v integrácii interaktívneho grafického rozhrania na ovládanie rôznych parametrov zvuku spolu s vizuálnymi efektmi, ktoré reprezentujú práve prehrávaný zvuk.

V prvých štyroch kapitolách je objasnený operačný systém Android, na ktorý bola aplikácia zameraná. Ďalej bol predstavený vizuálny programovací jazyk Pure Data, ktorý bol použitý na spracovanie granulárnej syntézy a zvukov pre kruhy. Vysvetlená bola taktiež platforma Plugdata, ktorá bola použitá na kompiláciu kompatibilného kódu pre Unity. V tretej kapitole bolo stručne vysvetlené z akých častí sa skladá Unity editor.

V piatej kapitole je už predstavený samotný návrh aplikácie, ktorý sa skladá z krátkeho úvodu kde je vysvetlená problematika pri výbere softvéru. Ďalej sú už vysvetlené jednotlivé prvky, ktoré boli v aplikácii použité. V spracovanom návrhu je opísaná integrácia Plugdata do Unity, granulárna syntéza pomocou Puredata, pridávanie a odstraňovanie kruhov v Unity, dotyk kruhu, vplyv pozície kruhu na zvuk, mixážny pult, automatické prehrávanie, granulárna syntéza, zvukové efekty a vizuálne efekty.

V rámci bakalárskej práce bola vypracovaná praktická realizácia aplikácie, ktorá popisuje implementáciu a problematiku s navrhnutým konceptom, od ktorého nastalo niekoľko zmien uvedených v šiestej kapitole. Taktiež bola táto kapitola doplnená o podkapitolu venovanú hudobným možnosťami a vymenovanými návrhmi na rozšírenie funkcií a stability aplikácie.

Literatúra

[1] Niall MOODY. *LibPdIntegration: A libpd wrapper for Unity*. online, 2025. Dostupné z: `https://github.com/LibPdIntegration/LibPdIntegration` [cit. 2025-11-19].

[2] Counterpoint Research. *Global Smartphone OS Market Share*. online, 2025. Dostupné z: `https://counterpointresearch.com/en/insights/global-smartphone-os-market-share/` [cit. 2025-11-09].

[3] Wikipedia. *Android (operačný systém)*. online, 2025. Dostupné z: `https://sk.wikipedia.org/wiki/Android_(opera%C4%8Dn%C3%BD_syst%C3%A9m)` [cit. 2025-11-10].

[4] Rajinder SINGH. *An overview of android operating system and its security. Int. J. Eng. Res. Appl*, 4(2):519–521, 2014.

[5] Android Developers. *Android Developers: Tools*. online, 2025. Dostupné z: `https://developer.android.com/tools` [cit. 2025-11-10].

[6] Valerio BRUSSANI. *ASVAAN: Semi-automatic side-channel analysis of Android NDK. arXiv preprint arXiv:2204.05911*, 2022.

[7] Android Developers. *Android NDK Guides*. online, 2025. Dostupné z: `https://developer.android.com/ndk/guides` [cit. 2025-11-10].

[8] Miller PUCKETTE. *Pure Data Documentation*. online, 2025. Dostupné z: `https://msp.ucsd.edu/Pd_documentation/` [cit. 2025-11-12].

[9] Timothy SCHOEN. *PlugData Documentation*. online, 2025. Dostupné z: `https://plugdata.org/documentation.html` [cit. 2025-11-10].

[10] Wasted Audio. *hvcc: The Heavy Compiler Collection*. online, 2025. Dostupné z: `https://github.com/Wasted-Audio/hvcc` [cit. 2025-11-13].

[11] Wasted-Audio. *Unsupported Vanilla Objects*. online, 2025. Dostupné z: `https://github.com/Wasted-Audio/hvcc/blob/develop/docs/10.unsupported_vanilla_objects.md` [cit. 2025-11-18].

[12] Wasted-Audio. *heavylib*. online, 2025. Dostupné z: `https://github.com/Wasted-Audio/heavylib` [cit. 2025-11-18].

[13] Wikipedia. *Unity (game engine)*. online, 2025. Dostupné z: `https://en.wikipedia.org/wiki/Unity_(game_engine)` [cit. 2025-11-18].

[14] Leonard J. PAUL. *Granulation of Sound in Video Games*. online, 2011. Dostupné na: https://ns2.videogameaudio.com/AES-Feb2011/AES41-04Feb11-GranulationOfSoundInVideoGames-LeonardJPaul.pdf [cit. 2025-12-04].

[15] Richard E. FLANAGAN. *FRACT OSC*. online, 2014. Dostupné z: https://www.richardeflanagan.com/fractosc [cit. 2025-11-19].

[16] yasirkula. *UnityNativeFilePicker: A native Unity plugin to import/export files from/to various document providers on Android & iOS*. online, 2025. Dostupné z: https://github.com/yasirkula/UnityNativeFilePicker [cit. 2026-05-19].

A Obsah elektronickej prílohy

Z dôvodu väčšej veľkosti súborov boli prílohy, teda presnejšie aplikácia a kompletný Unity projekt uložený na školský google drive účet na tomto odkaze `https://drive.google.com/drive/folders/1LsmgqaL5ZYC4FYLYWBEPy1bJVAgMpbXq?usp=drive_link`. K spusteniu aplikácie stačí stiahnuť `BakPraca.apk` súbor a nainštalovať ho na Android zaradení. K spusteniu Unity projektu je potreba mať nainštalovaný Unity Editor verzie 6000.2.12f1, pričom použitie inej verzie môže, ale nemusí spôsobiť chyby. Po otvorení projektu v Unity, sa môže editor prejavovať ako prázdny. V takom prípade je potrebné otvoriť súbor v zložke `Scenes` s názvom `Semestralka03.unity`.

- `/` — koreňový adresár priloženého archívu
  - `BakPraca.apk` — Hlavná aplikácia
  - `Unity.zip` — Unity projekt
    - `Assets`
      - `Scripts` — všetky kódy aplikácie
        - `Circle`
          - `addCircleUi.cs` — pridávanie kruhu
          - `CircleAudio.cs` — komunikácia s hvcc
          - `CircleBlink.cs` — vizuálna odozva
          - `CircleDrag.cs` — logika posúvania
          - `CircleRepeat.cs` — logika opakovania
          - `clickbang.cs` — hlavný kód
          - `removeCircleUi.cs` — odstraňovanie kruhov
        - `Grain`
          - `AddGrain.cs` — pridávanie zŕn
          - `AutoplayToggle.cs` — logika autoplay tlačidla
          - `BorderDrag.cs` — logika posúvania ohraničenia a zŕn
          - `BorderManager.cs` — logika aktívneho ohraničenia
          - `BorderPinch.cs` — zväčšovanie a zmenšovanie gestom
          - `BorderSpawner.cs` — pridávanie ohraničenia
          - `GrainAutoplay.cs` — pohyb zŕn
          - `GrainSample.cs` — načítanie a spracovanie zvukovej nahrávky
          - `RemoveBorder.cs` — odstraňovanie ohraničenia a jeho zŕn
          - `RemoveGrain.cs` — odstraňovanie zŕn
        - `Heavy`
          - `Wrapper.cs` — komunikácia s knižnicou hvcc
          - ... — a hvcc skripty
        - `Mixer`
          - `MixerSliders.cs` — logika posuvného prvku efektov
        - `PosOfTouch` — logika dotyku na obrazovke pre kruh
        - `WavFilePicker` — výber zvukovej nahrávky
    - `Scenes`
      - `Semestralka03.unity` — scéna v editore
    - ... — ďalšie súbory na správnu prácu projektu