



SZAKDOLGOZAT FELADAT

Mérnökinformatikus hallgató részére

Note Block hangminták felismerése zenefájlban

Hullámformátumú hangfájlokból a zenei hangok visszafejtése komoly probléma, erre az általános esetre létezik sok szoftver. Ezek többnyire wav/mp3 fájlokból generálnak MIDI fájlt, ami tartalmazza a megszólaló hangok időpillanatát és hangmagasságát. Ezen szakdolgozat feladat a problémának egy részhalmazával foglalkozik, jobb eredményt remélve. A Minecraft nevű játékban hallható Note Block mintákat tartalmazó hangfájlok felismerése a cél, a felismerés eredményét pedig egy erre a célra kifejlesztett szerkesztőprogram (Note Block Studio) saját formátumában bocsátjuk a felhasználó rendelkezésére.

Eddigi kereséseim alapján közvetlen konverzióra még nincs automatizált program / módszer. Leggyakrabban hallás után, manuálisan fejtik vissza a zenéket a felhasználók. Ez hosszadalmas, és jó fül kell hozzá. Automatizált módszer lehetne a hullámos fájlból MIDI készítése meglevő szoftverek segítségével, és a MIDI fájlt NBS-é alakítani, Note Block Studio-val. Ennek minden fázisa elég veszteséges és zajos, a gyakorlatban nemigen alkalmazzák.

A projekt célja egy olyan parancssori konvertáló program létrehozása, amely hullámformátumú (wav/mp3) zenából képes a hangmintákat felismerni, és ez alapján létrehozni egy Note Block Studio (NBS) fájlt. Alapvetően Note Block mintákat tartalmazó hangfájlok felismerése a cél, különböző időpillanatokban és hangmagasságokban, jó eredménnyel.

A hallgató feladatának a következőkre kell kiterjednie:

- Mutassa be az NBS formátum, és a Note Block-os zenék sajátosságait.
- Mutassa be az eddigi megoldásokat a problémára.
- Elemezze az automatizálás lehetőségét és az elérhető módszereket.
- Készítse egy parancssori programot, amely a konverziót el tudja végezni.
- Igazolja, hogy a konverzió a várthoz közeli eredményt ad.

Budapest, 2024. szeptember 23.

Dr. Charaf Hassan
egyetemi tanár
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

NOTE BLOCK HANGMINTÁK FELISMERÉSE ZENEFÁJLBAN

BUDAPEST, 2024

Tartalomjegyzék

1 Bevezetés.....	9
1.1 Zene ábrázolása, formátumok.....	9
1.2 Konverzió.....	10
1.3 Minecraft.....	10
1.3.1 Redstone, Note Block.....	10
1.4 Note Block Studio.....	11
1.5 Note Block-os zenék szubkultúrája.....	11
1.6 A dolgozat célja: Note Block-os zenék visszakottázása.....	12
1.7 Motiváció.....	12
1.8 A szakdolgozat további szerkezete.....	13
2 Irodalomkutatás.....	14
2.1 Irodalomkutatás hangfelismerés témaiban.....	14
2.1.1 Automatic Music Transcription: An Overview.....	14
2.1.2 From Audio to Music Notation.....	15
2.1.3 Unaligned Supervision for Automatic Music Transcription in The Wild.....	15
2.1.4 Omnidart: A General Toolbox for Automatic Music Transcription.....	16
2.1.5 Automatic Transcription of Polyphonic Vocal Music.....	18
2.1.6 Identification of Music Instruments from a Music Audio File.....	18
2.1.7 Automatic Note Recognition and Generation of MDL and MML using FFT.....	18
2.1.8 Algorithms for Non-negative Matrix Factorization.....	19
2.2 Hasonló alkotások bemutatása.....	20
2.2.1 Kereskedelmi szoftver a célra.....	20
2.2.2 Nyílt forráskódú szoftver a célra.....	20
2.2.3 Basic Pitch.....	20
2.2.4 MeloMIDI.....	23
2.2.5 Noteblock Music Yoinker.....	27
2.2.6 Galaxy Jukebox.....	28
2.2.7 NBSWave.....	29
2.3 Választott módszer, technológia.....	29
2.3.1 Felismerés módszere.....	29

2.3.2 Választott technológia.....	30
3 Tervezés.....	31
3.1 Technológiák ismertetése.....	31
3.1.1 Note Block-os zenék sajátosságai, formátuma.....	31
3.1.2 Spektrogram.....	32
3.1.3 Újramintavételezés.....	33
3.1.4 Optimalizációs algoritmusok.....	34
3.2 Követelmények.....	35
3.2.1 Funkcionális.....	35
3.2.2 Nem funkcionális.....	35
3.3 Rust könyvtárak bemutatása.....	35
3.3.1 Babycat.....	36
3.3.2 Spectrum analyzer.....	36
3.3.3 NBS fájlformátum-kezelő.....	36
3.3.4 Argmin.....	37
3.3.5 Aubio.....	37
3.4 Architektúra.....	37
3.4.1 Áttekintés.....	38
3.4.2 Egy ütem felismerése.....	38
4 Önálló munka bemutatása.....	40
4.1 A Rust nyelv.....	40
4.2 Modulok bemutatása.....	41
4.2.1 Complex spectrum, Observer.....	41
4.2.2 Wave.....	42
4.2.3 Fourier.....	43
4.2.4 Debug.....	44
4.2.5 Note.....	45
4.2.6 Nbs.....	46
4.2.7 Tempo.....	48
4.2.8 Optimize.....	48
4.2.9 Main.....	54
5 Önálló munka értékelése, eredmények.....	56
5.1 Értékelés módszere.....	56
5.2 Hangfelismerés pontossága.....	56

5.2.1 Tesztkörnyezet.....	56
5.2.2 Változtatható paraméterek ismertetése.....	58
5.2.3 Alapdob.....	59
5.2.4 A Nelder-Mead módszer és a Steepest Descent összehasonlítása.....	60
5.2.5 Tesztelés a teljes adathalmazon.....	61
5.2.6 Algoritmus, kezdeti érték, iterációszám.....	63
5.2.7 Algoritmusok grafikonos összehasonlítása.....	65
5.3 Tempófelismerés pontossága.....	66
5.4 Összehasonlítás eddig elérhető módszerekkel.....	67
5.5 Robusztusság.....	69
6 Összefoglaló.....	70
7 Irodalomjegyzék.....	71
8 Függelék.....	74

HALLGATÓI NYILATKOZAT

Alulírott szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettettem, egyértelműen, a forrás megadásával megjelöltetem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2024. 12. 05

.....

Összefoglaló

Hullámformátumú hangfájlok ból a zenei hangok visszafejtése (Automatic Music Transcription, AMT) nehéz feladat, erre az általános esetre létezik sok szoftver. Ezek többnyire wav/mp3 fájlok ból generálnak MIDI fájlt, ami tartalmazza a megszólaló hangok időpillanatát és hangmagasságát.

Ezen szakdolgozat a problémának egy részhalmazával foglalkozik, megmutatva, hogy a specializációval jobb eredmény érhető el. A Minecraft nevű játékban hallható Note Block mintákat tartalmazó hangfájlok felismerése a cél, a felismerés eredményét pedig egy erre a cérla kifejlesztett szerkesztőprogram (Note Block Studio) saját formátumában bocsátjuk a felhasználó rendelkezésére.

A feladat elvégzésére készíték egy parancssori programot, amely időpillanatonként elemezve végigmegy a zenén, és kiadja a kért átiratot. Egy időpillanatban képezi az eredeti zene, illetve a hangminták spektrogramjait, és egy erre a cérla létrehozott költségfüggvény, illetve a gradiense segítségével megkeresi a minimumhelyet, azaz azt, hogy melyik hangnak milyen hangerőn kell szólnia, hogy az eredeti zenéhez a lehető legközelebb kerüljön.

A programot egy 9,5 órányi zenét tartalmazó adathalmazon teszteltem, 4 különböző hangszeret ismert fel többszólamú módon. Az nbswave nevű, Note Block Studio fájlt hullámformává szintetizálni képes program kimenetén a program 1,6%-os négyzetes hangerőhibát (mean squared error) ért el a csendéhez viszonyítva, a teljes adathalmazon.

Az elért eredmény mutatja, hogy a formátum sajátosságait kihasználva sokkal nagyobb pontosság érhető el, mint általános módszereket alkalmazva.

Abstract

Transcribing waveform audio files into notes (Automatic Music Transcription, AMT) is a challenging task, and numerous software solutions exist for this general case. These typically convert wav/mp3 files into MIDI files, which include the timing and pitch of the notes played.

This thesis addresses a subset of the problem, demonstrating that specialization can achieve better results. The goal is to recognize audio files containing Note Block sound samples from the game Minecraft and provide the recognized output in the format of a dedicated editing tool, Note Block Studio.

To accomplish this task, I developed a command-line program that processes the music by analyzing it moment by moment and generates the desired transcription. At each moment in time, the program generates spectrograms of both the original music and the sound samples, then uses a custom cost function and its gradient to find the minimum, determining which sounds should play at what volume to approximate the original music as closely as possible.

The program was tested on a dataset containing 9.5 hours of music, identifying four different instruments polyphonically. When evaluated on the output of a program called nbswave — which synthesizes Note Block Studio files into waveforms — the program achieved a mean squared volume error of 1.6% relative to that of silence across the entire dataset.

The results show that leveraging the specific characteristics of the format can yield significantly higher accuracy compared to general-purpose methods.

1 Bevezetés

A zene régóta fontos szerepet tölt be az emberek életében, a mai technológiákkal pedig sok minden jóval könnyebben elérhető, mint régebben. Manapság könnyen és ingyen meghallgathatunk Beethoven műveket YouTube-ról[1], vagy kereshetünk hozzájuk kottákat, ha le szeretnénk játszani őket[2]. Ezen lehetőségek mögött komoly technológia van. Csak néhányat említve: hogyan lehet eltárolni ezt az adatot egy számítógépen, vagy hogyan lehet ezt az adatot a szerverről elküldeni a kliensnek, úgy, hogy az valós időben le tudja játszani. Ezért az eltárolt formátum is lehet többféle, amik alapvetően különbözhetsznek.

1.1 Zene ábrázolása, formátumok

Egy zenét eltárolni ugyanis többféleképpen is lehet: alapvetően más adatokat tárolunk egy hullámförmatumú hangfájlban, mint egy Musical Instrument Digital Interface (MIDI)[3]-szerű fájlban. Az analóg világból olyan példát tudok hozni, mint a bakelitlemez és egy papírkotta közötti különbség. Az előbbi eltárolja a hanghullámokat, amit egy gramofon, vagy egy hangszóró vissza tud könnyen játszani, akár egy teljes zenekart igénylő darab esetén is. A kotta lejátszása ezzel szemben egy teljes zenekart igényelt régen. A digitalizált verzió lejátszása könnyen megy, bár nehéz úgy megcsinálni, hogy valósághűen szóljon.

Ugyanakkor, hogyha egy zongorán egy kézzel könnyen lejátszható verziót szeretnénk csinálni a bakelitlemezből, az elég nehézkes, és például hallás után kell megtippelni az egyes hangokat. A kotta egyszerűsítése viszont sokkal könnyebb, elég összevágni a fő dallamot tartalmazó részeket.

Ehhez hasonló a digitális világban a hullámszín hangfájlok, például: wav (Waveform Audio File Format)[4] vagy mp3 (MPEG-1/2 Audio Layer III)[5], és a MIDI vagy MusicXML (Music Extensible Markup Language)[6] közötti különbség. Az előbbiekből másodpercenként például 44100 darab számérték van tárolva (esetleg tömörítve, hogy kevesebb helyet foglaljon), ami egy, ha úgy tetszik, függvény értékeit mondja meg az adott időpillanatokban. Ezt fel lehet könnyen venni egy mikrofonnal, és visszajátszani egy hangszóróval, akár zenét, akár bármiféle más, ember által hallható hangot, beszédet, zajt. Ezzel szemben a kottával analóg tárolás esetén adott zenei hangok kezdőidőpontja, hangszere, hangereje és hossza van eltárolva. Ezt könnyen meg lehet jeleníteni, időben pontos zenék

esetén kottaként, de legalábbis lehet pontosan tudni, hogy melyik időpillanatban milyen zenei hang szól.

1.2 Konverzió

A hallott zene lekottázása régóta fontos kérdés. Ennek a számítógépes megfelelője a hullámformátumú fájlból a szóló hangok időpillanatának, hangszerének, hangmagasságának és hangerejének meghatározása. Ez kifejezetten trükkös kérdés, sok szoftver született rá, általános, és különböző specifikusabb esetekre, amelyek többé-kevésbé automata módon tudnak jobb-rosszabb eredményt elérni. Többre részletesebben kitérek a 2.1, illetve 2.2 fejezetekben. Ezen szakdolgozat is egy specifikus esettel foglalkozik, amire még nem próbáltak automata konverziós szoftvert alkotni. Így ezen téma a továbbiakban részletesen ki lesz fejtve.

1.3 Minecraft

A specifikus eset bemutatásához először szükséges egy számítógépes játékot ismertetnem, a Minecraftot. Pontosabban valószínűtlen, hogy az olvasó nem hallott róla, mivel jelenleg a legtöbb példányban elkelt videójáték a világon, 300 millió darab példánnyal[7], de azért röviden ismertetem. A játék egy 3 dimenziós, kockákból álló homokozó (sandbox) típusú játék, ahol a játékos nagyon sok módon tud kreatív lenni. Sokféle építőelem, kocka (blokk) áll rendelkezésre, amiket szinte bármilyen formációban le tud tenni a világban, ami így egy üres 3 dimenziós, kockákra osztott koordinátarendszerként is felfogható.

1.3.1 Redstone, Note Block

A játékon belül van egy áramkörökhöz hasonlítható, vöröskőnek (redstone-nak) nevezett alrendszer, amivel egészen komplex logikákat lehet megvalósítani nagyon egyszerű alkotóelemekből. Korábban készítettem például egy 99x99 pixels grafikus függvénymegjelenítőt[8], többek között. Az egyik ilyen áramköri elemmel, a hangdobozzával[9] (Note Block) lehet hangokat kiadni a játékon belül. Én a Note Block megnevezést fogom használni, mert azt szoktam meg, és a magyar fordítás nagyon idegennek hangzik.

1.4 Note Block Studio

Note Block-os zenék írására készítettek egy külön stúdióprogramot is, a Note Block Studio-t. A fejlesztője régebben abbahagyta a fejlesztését, majd mások nyílt forráskódú szoftverként felkarolták, így a ma használt verzió az Open Note Block Studio (ONBS) [10]. Ebben a játék használata nélkül is lehet zenéket tervezni és visszahallgatni, a Note Block által kiadott hangok felhasználásával. Egy kompozíciót el lehet menteni a stúdió saját fájlformátumában, NBS-ben[11], és lehet exportálni többek között mp3 (hullámformátumú) hangfájlként, illetve a játékban a zenét lejátszó vöröskőáramkörként is.

1.5 Note Block-os zenék szubkultúrája

A Minecraftnak egy jelentős és fontos részét teszik ki a szerverek, ahol több játékos játszhat együtt. A játék fejlesztője, Mojang, nem támogatja expliciten, és nem működik szorosan együtt a legnagyobb szerverek üzemeltetőivel, fejlesztőivel, így gyakori az, hogy egy régi játékverzióval jobb a játékélmény, mint egy újjal. Például, a legtöbb online játékossal bíró szerver, ahol ebben a pillanatban 33000-en játszanak [12], a Hypixel [13], még mindig 1.8.9-et javasol, mint a legtámogatottabb verzió, pedig azt 2015 decemberében adták ki, majdnem 9 éve. Ennek ellenére nagyon sokan játszanak ilyen harmadik féltől származó szervereken, ezeknek külön-külön is kialakultak szubkultúrái.

A Hypixelen, az egyik legnépszerűbb szerveren, sokféle különböző minijáték közül választhat az ember. Egy másik, szépen kidolgozott szerver a Wynncraft nevű MMORPG (massively multiplayer online role-playing game, nagyon sok szereplős online szerepjáték), ahol izgalmas küldetések, komoly szintrendszer, óriási terep és sok más várja a játékosokat [14]. Mindkét szerveren, és más szervereken is gyakran felhasználnak háttérzeneként Note Block-okból összeállított zenéket, mivel azt lehet könnyen lejátszani a játékos módosítatlan kliensén. Ilyen zenékből pár példa a Hypixel Skyblock minijátékához tartozó lejátszási lista, ahol több zenének is 100 000 fölötti a megtekintése YouTube-on [15], illetve a Wynncraft OST (official soundtrack, hivatalos játékháttérzene), ahol több, mint 200 zenét írtak a játékhoz [16]. Ezenkívül sokan hobbiból írnak saját Note Block-os zenéket, vagy írnak át egyéb kompozíciókat úgy, hogy Note Block-akkal lejátszható legyen.

1.6 A dolgozat célja: Note Block-os zenék visszakottázása

A fentebb említett MIDI fájlhoz tehát hasonló az NBS (Note Block Studio) fájl olyan szempontból, hogy a megszólaló hangok időpontjai, hangmagasságai, és hangerejei vannak benne eltárolva, de specifikusabb abból a szempontból, hogy ismerjük a konkrét Note Block hangmintákat, amik szerepelnek a fájlban (a hullámformájuk fix), és egyéb paraméterek is kötöttebbek. A hangmagasság normál esetben 2 oktávon belül van egy-egy hangszer esetén, a hangszerek mennyisége is korlátozott (5 és 16 között, verziófüggő), a lecsengés pedig fix, ismert, a játék egyetlen konkrét hangmintát (hullámformájú hangszerhangot) torzít, és játszik le teljesen végig. Ezeket nem figyelembe véve az általános konvertáló programok értékes információt nem használnak fel, amivel egy specifikus konvertáló program jobb eredményt tudna elérni.

Eddigi kereséseim alapján közvetlen konverzióra még nincs automatizált program / módszer. Leggyakrabban hallás után, manuálisan fejtik vissza a zenéket a felhasználók. Ez hosszadalmas, és jó fül kell hozzá. Automatizált módszer lehetne a hullámos fájlból MIDI készítése meglevő szoftverek segítségével, és a MIDI fájlt NBS-é alakítani, Note Block Studio-val. Ennek minden fázisa elég veszteséges és zajos, a gyakorlatban nemigen alkalmazzák.

A projekt célja egy olyan parancsos konvertáló program létrehozása, amely hullámformátumú (wav/mp3) zenából képes a hangmintákat felismerni, és ez alapján létrehozni egy NBS fájlt. Alapvetően Note Block mintákat tartalmazó hangfájlok felismerése a cél, különböző időpillanatokban és hangmagasságokban, jó eredménnyel.

1.7 Motiváció

Ezen átírás több szempontból előnyös lenne, például a hangok ismeretében lehet vizualizációt¹ készíteni a zenéről, amit a hangok nélkül, csak a hullámos hangfájl ismeretében nem lehetne. Egy másik felhasználási lehetőség, ha valaki át szeretné dolgozni a Note Block-os zenét², újrahangszerelni. Ehhez is kell tudni, hogy milyen hangok szólnak mely időpillanatokban. Még egy felhasználási lehetőséget célszerű megemlíteni, ez pedig a kotta készítése, ugyanis ha a megszólaló Note Block-ok ismertek, abból már viszonylag könnyen

¹<https://www.youtube.com/watch?v=L7TTUkqprQ0>

²https://www.youtube.com/playlist?list=PLuxIgMW_nasep2O39FE5GZay89wxcGhrL

lehet kottát készíteni. Ennek segítségével utána meg lehet tanulni például a zenét hangszeren lejátszani, rekreációs céllal.

1.8 A szakdolgozat további szerkezete

A továbbiakban tanulmányozom a téma szakirodalmát, bemutatok hasonló, és kapcsolódó alkotásokat, illetve kiválasztom a használt módszert és technológiát. Ezután ismertetek pár fontos módszert, elméletet, amit fel fogok használni a programban. Elemzem a követelményeket, bemutatom a felhasznált könyvtárakat és a tervezett architektúrát. Az önálló munka bemutatása című fejezet a kódról szól, hogy pontosan hogyan történik a felismerés. A többi fejezetben is jelentős mennyiségű önálló munka bemutatása zajlik, például az önálló munka értékelése során sok paraméter változtatásával megvizsgálom a felismerés pontosságát, és összehasonlítom az eddig elérhető módszerekkel. Végül összefoglalom a szakdolgozatot, és fejlesztési javaslatokat teszek.

2 Irodalomkutatás

Ebben a fejezetben a témához kapcsolódó szakirodalmat keresek, elemzek, és felmériem a rendelkezésre álló módszereket. Összegyűjtök továbbá hasonló programokat, végül vázolom a választott technológiát és módszert.

2.1 Irodalomkutatás hangfelismerés témaiban

Kifejezetten Note Block mintafelismerés témaiban nem találtam sem cikket, sem programot. Minecraft és zene keresztmetszeteként több cikket is találtam, az egyikben felhasználták a 3D teret zeneművek tanítására és tanulására, illetve kottázásra[17], egy másikban pedig Note Block-okat használtak az általános iskolások, hogy dallamokat alkossanak[18]. Viszont ezek egyike sem kapcsolódik szorosan a témához, így inkább az általános eset, az automata zeneátírás (Automatic Music Transcription, AMT) irányában kerestem cikkeket.

2.1.1 Automatic Music Transcription: An Overview

Ez a cikk[19] felvázolja, összefoglalja a témakört, hasznos betekintést nyújt. Általában ismerteti először a problémát, felvázolja a felhasználási lehetőségeket és többek között a főbb nehézségeket. Bemutatja, milyen különböző technikákat, módszereket alkalmaznak a probléma megoldására. Ezek közül a két manapság legelterjedtebbet, a Non-negative Matrix Factorization-t (NMF) és a neurális hálókat (NN) részletesen is bemutatja. Végül kitér különböző kapcsolódó kérdésekre, ezek közül nekem a Context-Specific Transcription a releváns: ez jelenti azt, hogy ismerjük előre a megszólaló hangot. Továbbá fontos kérdés az ütős hangszerek felismerése, ez azonban a Note Block-os zenék esetében nem sokban különül el a dallamhangszerektől, mivel a Minecraft ezeket egységesen kezeli, és az ütős hangszereknek is ugyanúgy lehet állítani a hangmagasságát, és a ritmusa sem lehet komplexebb, gyorsabb, mint a dallamhangszereké.

A cikkben a két főbb megoldási módszer (a Non-negative Matrix Factorization és a neurális hálók) összehasonlítását érdemes egybevetni az általam megoldandó problémával. Először is a cikk kihangsúlyozta, hogy minden módszer releváns, és bizonyos körlémények között jobb megoldást tud produkálni, mint a másik. Az első hátrány, amit az NMF ellen

említettek az az, hogy a minta (pl. a C4 hanghoz tartozó) spektrumok lineáris kombinációja nem biztos, hogy jól kiadja a felvételben található hangot, míg ha több C4 hanghoz tartozó spektrumot tesznek be mintaként, akkor pedig a lineáris kombinációk terében túl sok lesz az invalid spektrum. Vegyük észre viszont, hogy ez a Note Block-os hangokra sokkal kevésbé vonatkozik, mivel a hangok és a spektrumaik pontosan ismertek, és nem kell számolni a valóságos felvétel során történő torzulással. A neurális hálók egyik hátrányaként pedig felsorolták, hogy nincs elég adat (vagy nem elég diverz) a betanításhoz (ez probléma lehetne számonra is, mivel ugyan van egy nagy adathalmazom, de nem elég diverz a cél eléréséhez). A fontosabb hátrány viszont, hogy a cikk (és egyéb hivatkozott cikkek) szerint az NMF-fel szemben a neurális hálókat nem lehet pár másodperces minta-hanggal jóval precízebbé tenni – azaz nincs egyértelmű lehetőség a konkrét, ismert hangminták felhasználására a felismerés gyorsítása és pontossása érdekében. Mivel ez a fő indok arra, hogy a Note Block minták felismerésének speciális esete miért könnyebb, mint az általános eset, ezért, feltéve, hogy más szakirodalom másként nem vélekedik, a neurális hálók módszerének elemzése a probléma megoldására ezen szakdolgozat keretein túlnyúlik.

2.1.2 From Audio to Music Notation

Ezen cikk[20] két évvel fiatalabb, mint az előző (ez 2021-es), és ugyanúgy bemutatja a problémát, de jobban ráfókuszál a neurális hálókra. Ennek megfelelően részletesebben taglalja az elérhető adathalmazokat, illetve a pontosság kiértékelésének lehetséges módszerét is. Részletesebben bemutatja egy neurális háló felépítését, illetve bemutatja a közvetlen kotta generálásának módszerét is, amely több információt tartalmaz, mint egy sima MIDI kimenet (előjegyzés, ütem, pontosabb ritmusok). Végül felsorolja a megoldás nehézségeit. Viszont ebben a cikkben nem találtam említést a pontos hangminták felhasználásának lehetőségéről, így továbbra is előnyben részesítek más (nem NN) módszereket.

2.1.3 Unaligned Supervision for Automatic Music Transcription in The Wild

Ezen cikk[21] 2022-es, és egy olyan módszert mutat be, ami általános, töbhangszeres, többszólamú zenét képes felismeni, a korábbi, általuk hivatkozott programoknál jobb eredménnyel. Ez is gépi tanulást használ a háttérben, és hangfájlokkal és MIDI fájlokkal lehet betanítani. Nem kell viszont, hogy pontosan össze legyenek illesztve

időben (innen az unaligned jelző), azt a rendszer automatikusan elvégzi. Ez sok tekintetben könnyíti többek között az adathalmaz beszerzését.

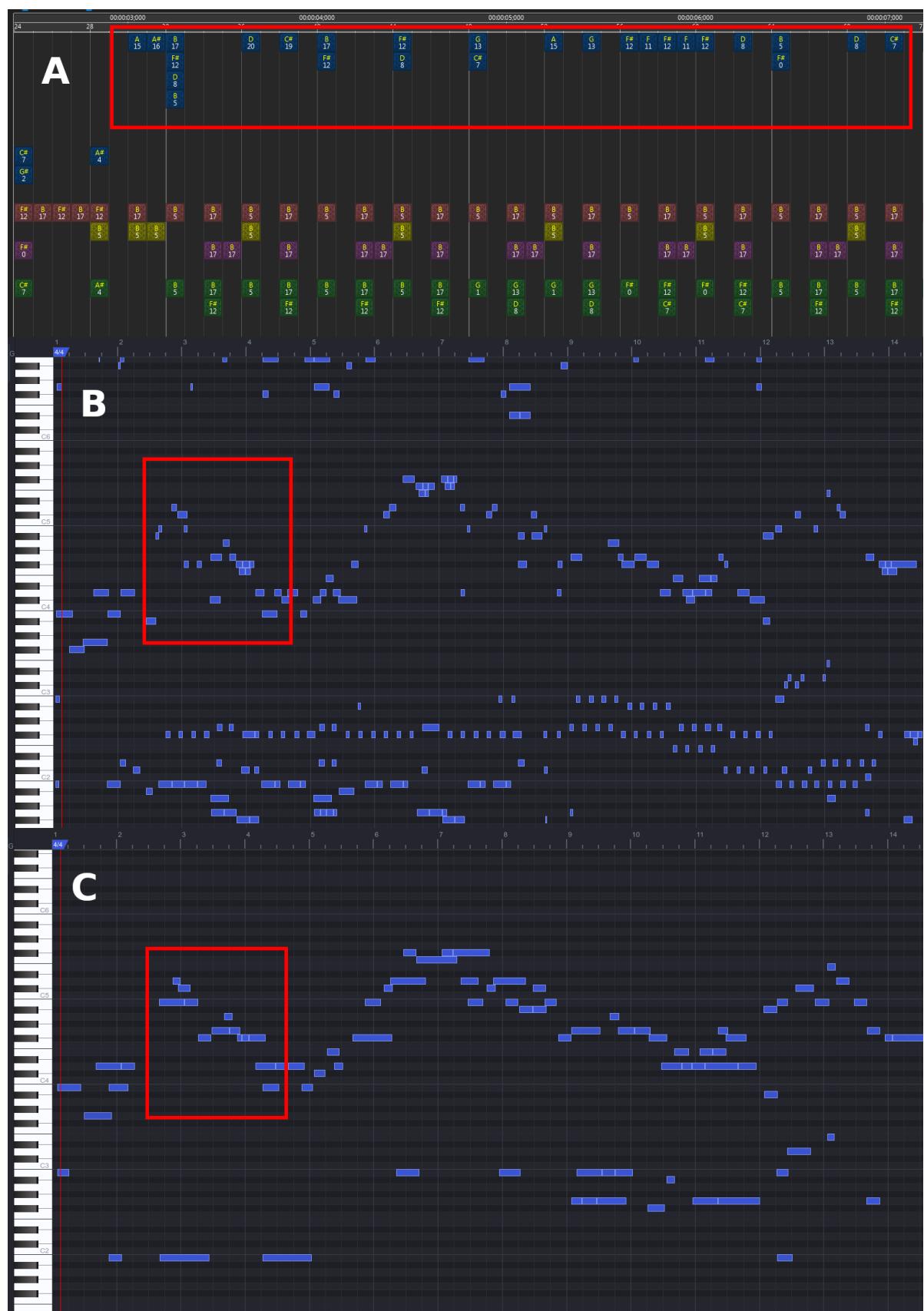
2.1.4 Omnidart: A General Toolbox for Automatic Music Transcription

Ez már egy gyakorlatiasabb projekt[22], egy Python könyvtárról szól (omnidart), amivel a felhasználók könnyen képesek automatikusan zenét átírni, akár egy parancssal. Ez az első olyan eszköztár, ami átfogó eszközöket ad, adathalmaz letöltésére, betanításra, egy- és többszólamú zene, dob, illetve énekhang átírására, többek között. Ez más cikkek módszereit gyűjti össze, és teszi könnyen felhasználhatóvá. A háttérben Tensorflow-t használ, ami egy összetett szoftverkönyvtár gépi tanuláshoz. Általános, többhangszeres zenénél a teszt adathalmazon 67%-os pontosságot ért el, ami az én célomra nem biztos, hogy elegendően pontos lesz.

Mivel a program könnyen elérhető és kipróbálható, leteszteltem (ábra 1) egy Note Block-os zenén. Finomhangolás nélkül nem várható el a tökéletesség, és egy újból betanítás illetve egyéb opciókkal való kísérletezés valószínűleg segítene a felismerésen. A módszerből adódóan viszont itt sem tudnánk felhasználni a hangminták pontos ismeretét.

Az 1. ábrán bekereteztem a fő dallamot, amely felismerhető minden fajta átíráson. Az összes alapértelmezetten elérhető opció közül ez a kettő adott releváns eredményt, ezek közül is a piano adott láthatóan pontosabb átírást. Az sem tökéletes: egyrészt nem veszi figyelembe (nem is tudja) a különböző hangszereket, másrészt van több magas hang a felismerésben, ami eredetileg nem volt ott, illetve van több olyan akkord (például rögtön a harmadik hanghalmaz a bekeretezett részben), ahol a mélyebb, egyszerre szóló hangokat nem találta meg. A MIDI ábráit a <https://signal.vercel.app/> programmal készítettem, a felhasznált zene pedig az Eight-Legged Foe Frigiduck-tól[23].

Ami még fontos kérdés, érdemes közelebbről megnézni: a beat tracking, avagy a zene ütemének felismerése. Az Omnidart újít ebből a szempontból az elérhető könyvtárakon (mint például a madmom, vagy a librosa), mivel eddig hullámformátumú hangfájlon futtattak algoritmust, az Omnidart pedig szimbolikus zenei adaton (MIDI-szerű adaton) futtat, az ütem felismerésének céljából. Mivel nekem előbb kell majd felismernem az érdekes időpillanatokat, mielőtt bármilyen szimbolikus adatom lenne, az előbbi módszer lesz fontos. A Note Block-os zenékben nincs tempótájolás, és ütemtől való apróbb elcsúszások is csak pontatlanság miatt adódhannak.



Ábra 1: Az Omnidart program által adott MIDI-ről

A: az eredeti NBS fájl OpenNBS-sel megnyitva

B, C: omnidart piano, illetve notestream átírás eredményeként kapott MIDI

2.1.5 Automatic Transcription of Polyphonic Vocal Music

Ez a cikk[24] többszólamú énekhang átírásával foglalkozik. Spektrogramot készít a forrásból, szeparálja a szólamokat, és egy másik modellt is integrál, hogy a több szólamot fel tudja ismerni. Bekategorizálja szoprán, alt, tenor és basszus (SATB) típusokba, rejtett Markov modell felhasználásával. Kiértékelték a módszert, és elérte az elvárt, state-of-the-art eredményeket.

2.1.6 Identification of Music Instruments from a Music Audio File

Ez a cikk[25] két egyszerre, ugyanazon a hangmagasságon szóló, különböző hangszer felismerésével foglalkozott. 3 különböző hangszer 12 hangmagasságából választottak kettőt, és azt ismerték fel. Két része van a rendszernek: először a jellemzőket kiszedték a mintából, majd osztályozták őket. Az osztályzáshoz neurális hálókat, a jellemzők kiszedésére mel-frekvencia cepstrumot használtak. Ehhez először egy ablakfüggvényt kell alkalmazni (ők a Hamming-ablakot választották), majd Fourier-transzformációt kell végrehajtani (Fast Fourier Transform, FFT), ezután a mel skálára áttérés érdekében háromszög alakú szűrőt és diszkrét koszinusz transzformációt alkalmaztak. A projekt sikeres volt, a különböző hangszereket eredményesen sikerült felismerni.

2.1.7 Automatic Note Recognition and Generation of MDL and MML using FFT

Ezen cikk[26] egy saját formátumot használ MIDI helyett, lényegileg hasonló céllal, ez az MDL és MML. Ezeket korábbi cikkekben bemutatták részletesebben, de ezekben is hang kezdési időpillanatok, hangerők, hangmagasságok és hanghosszak vannak eltárolva. Ennek a visszafejtése hullámformátumú hangfájlból volt a feladat. Komolyan, viszonylag részletesen taglalják a képleteket, Fast Fourier-transzformációt használnak, viszont tesztelés gyanánt egyetlen 15 másodperces, egyszólamú, szinusz-hullámokból álló zenét próbáltak ki. Így nem komoly programról van szó, csak egy módszerrel való kísérletezésről.

Mégis az, hogy egy saját, egyszerűsített formátumot használnak, amit konvertált hangfájlból visszafejteni szeretnének, sok tekintetben hasonlít az én problémámra.

2.1.8 Algorithms for Non-negative Matrix Factorization

A cikk [27] a nemnegatív mátrixfelbontásról ír, ezt a 2.1.1-ben taglalt cikk javasolta, mint egy lehetséges módszer. Nem specifikusan zenéről szól, hanem a mögöttes matematikáját fejti ki jobban a problémának. Adott egy V nemnegatív mátrix, amit egy W és H , szintén nemnegatív mátrixok szorzataként szeretnénk minél jobban közelíteni. A cikk két módszert ismertet, amik bizonyíthatóan lokális minimumhoz konvergálnak.

Ha zenére használjuk fel a módszert, akkor [19] alapján intuitív jelentése is van a mátrixoknak. Az egyik tartalmazza a lehetséges hangminták spektrumait, azaz, hogy milyen hangmagasságon, frekvencián szól a hang, illetve a felhangjai. A másik mátrix pedig azt mutatja meg, hogy időben az előbbi mátrix mely spektrumaiból milyen lineáris kombinációval áll elő a szorzatmátrix azon oszlopa, amelyik ehhez az időpillanathoz tartozik, azaz mikor milyen hangok szólnak.

Ezzel a módszerrel el lehetne készíteni egy olyan mátrixot a hangmintákból, ami az én esetemre specifikus. Amit viszont nem lehet, legalábbis a módszer módosítása nélkül figyelembe venni, az a hangok lecsengése. Mivel a Note Block-os minták lecsengése is egyértelmű, és jellemző a hangszerre, ezzel a felismerés pontosságát elősegítő információt veszítünk. A másik pedig a ritmus figyelembevétele, amennyiben korábban meg tudom határozni a fontos időpillanatokat, a többi időpillanatbeli felismerést is az ütem szempontjából érdemes vizsgálni. Azaz az adott ütem felismeréséhez érdemes felhasználni az egész időintervallumot, ami a következő ütemig tart. Ennek az az oka, hogy hang csak az ütem kezdőpillanatában fog megszólalni, és onnantól egyértelmű a folytatása. Emiatt ha a modellünknek megengedjük, hogy ütem közbeni hangok felvételét/leállítását is jelezni tudja, az nem fogja a zenét a lehető leg pontosabban modellezni. Valami hasonló, minimumkereső módszer ellenben célra vezethet, és azzal meg lehet oldani, hogy csak az adott időpillanatban kombinálja lineárisan a hangokat. Az NMF viszont használható lehetne egyfajta előfelismeréshez, állapottér-csökkentéshez, hogy a későbbi minimumkereséskor ne az összes hang lineáris kombinációja között keresse a megoldást, hanem csak a potenciálisan szóló hangok terében.

2.2 Hasonló alkotások bemutatása

2.2.1 Kereskedelmi szoftver a célra

Sokféle kereskedelmi szoftver is létezik zeneátírásra [19], többek között Melodyne³, AudioScore (Sibelius)⁴, ScoreCloud⁵ és AnthemScore⁶. Ezek többsége elég drága, és feltehetően más-más célra jók. Az AnthemScore-t kipróbáltam, és egy-egy zenére, amit adtam neki, elég jó, tiszta kimenetet adott. Egy-egy hang kimarad persze, de lehet állítani, hogy hány hangot tegyen be, és minden a legvalószínűbbet teszi be. Ezeket nagyon hasznos, és érdekes lenne közelebbről megvizsgálni, milyen pontosak, milyen extra képességeik vannak, mennyire tudnak különböző fajta zenéket felismerni. A szoftvert magát persze úgysem tudnám felhasználni a felismeréshez.

2.2.2 Nyílt forráskódú szoftver a célra

A probléma fontossága miatt nyílt forráskódú szoftver is akad bőven, ami hullámformátumú hangfájlból MIDI-t tud generálni. Itt⁷ található egy lista, ahol sok szoftvert, projektet felsorolnak. Ezek közül említésre méltó a WaoN, ami egy régi program, de régebben többször használtam teszt jelleggel. Eléggé zajos a kimenete. Fontos még megemlíteni a NeuralNote-ot⁸, ami egy digitális hang munkaállomásba (DAW, Digital Audio Workstation) integrálható plugin, ami a háttérben a Spotify által nyílt forráskódúvá tett basic-pitch nevű, AMT könyvtárat használja.

2.2.3 Basic Pitch

A Basic Pitch-hez tartozik egy cikk is [28], és egy weboldal is, ahol könnyen fel lehet tölteni egy hullámformátumú hangfájlt, módosítani a paramétereket, és letölteni a hozzá tartozó MIDI fájlt. Kiemelkedően fontos manapság, hogy a szoftvert könnyen felfedezhetővé

³<https://www.celemony.com/en/melodyne/what-is-melodyne>

⁴<http://www.sibelius.com/products/audioscore/ultimate.html>

⁵<https://scorecloud.com/>

⁶<https://www.lunaverus.com/>

⁷<https://gist.github.com/natowi/d26c7e97443ec97e8032fb7e7596f0b0>

⁸<https://github.com/DamRsn/NeuralNote>

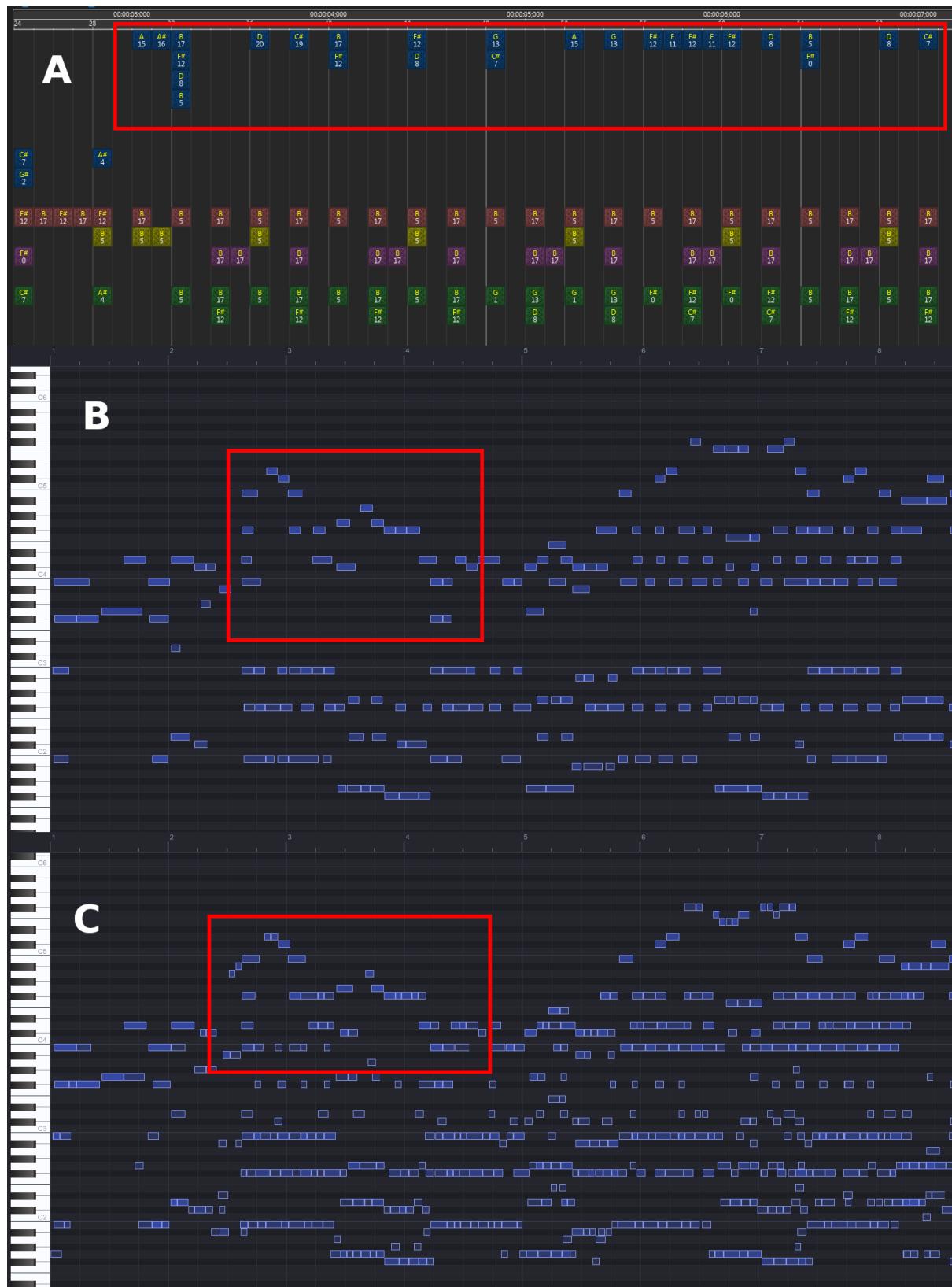
és kipróbálhatóvá tegyék, egyéb esetben nemigen fogja megragadni az emberek figyelmét, még akkor sem, ha egyébként egy technológiai szempontból úttörő dologról van szó.

A Basic Pitch egy hangszerfüggetlen, alacsony erőforrás-felhasználású, neurális háló alapú modell, ami jó felismerést tud biztosítani, és a legjobb, specializált AMT rendszerekhez képest is csak kicsit marad el. [28] Ugyan többféle hangszert is fel tud ismerni, a hangszerek zenénként különböztek a cikkükben, és fejlesztési lehetőségként sorolták fel az egyszerre szóló többféle hangszer felismerését. Ugyanakkor a GitHub oldalukon⁹ írták, hogy cikk írása óta fejlesztettek a programon, és valóban: a cikkben fejlesztési lehetőségként felsorolt hajlításfelismerés a NeuralNote felületén már elérhető. Érdekes még megjegyezni, hogy a fentebb említett Melodyne nevű kereskedelmi szoftver pontosságával vetekszik a zongorafelismerésben.

Mivel ez a program is könnyen kipróbálható, itt is végzek egy hasonló tesztelést (ábra 2), mint az Omnidart esetében (ábra 1). Az eredmény hasonló, a zene teljesen felismerhető, de nem annyira pontos. Egy-két apró díszítőhang hiányzik belőle, van pár hang, ami igazából nem volt ott, és a hangszerek közötti különbségtétel itt sincs meg. Kétféle beállítással próbáltam: a B ábra az alapértelmezett értékekkel (hang-szétbontás, hangok száma / magabiztosság, minimum hanghossz), míg a C ábra erősebb szétbontással, több hanggal és kisebb minimum hosszal történő konverzió eredményét mutatja. A C ábra módszere így az apróbb részleteket jobban felismeri, viszont több olyan hang is belekerül, aminek nem kellene ott lennie. Ez a kompromisszum, amit meg kell hozni ilyen esetben.

A probléma ugyanaz, mint az Omnidart, és bármilyen más, általánosabb célú szoftver esetén, hogy az erősségük az általanosságban rejlik. Sokféle zenét elég jól fel tudnak ismerni. A Note Block-os zenék felismeréséhez viszont a már sokszor említett specifikus tulajdonságok sokat segítenek, ami egy ilyen általános célú szoftver esetén elveszne, amennyiben ilyen könyvtárat, vagy módszert használnék a felismeréshez.

⁹<https://github.com/spotify/basic-pitch>



Ábra 2: A Basic Pitch program által adott MIDI-ről

A: az eredeti NBS fájl OpenNBS-sel megnyitva

B, C: az átírás eredményeként kapott MIDI, különböző beállításokkal

2.2.4 MeloMIDI

A MeloMIDI¹⁰ egy saját programom, amit korábban írtam. Nagyon sok témába illő tapasztalatom és ismeretem az akkori utánaolvasásaim, illetve a program megírásának az eredménye. Ez egy, a fentebb bemutatott programokhoz hasonló, általános AMT megoldás, persze nem annyira kifinomult, mint a fentebbi programok.

Ennek ellenére van egy grafikus felhasználói felülete, ahol be lehet tölteni egy hullámformátumú hangfájlt, lehet állítani a konverzió különböző paramétereit. A program ezután elkészíti a zenefájl spektrogramját (amiről a 3.1.2 fejezetben is lesz szó), majd a spektrogram és a beállítások alapján megkísérel zenehangokat felismerni. Ezeket azután kirajzolja a felhasználói felületre, a spektrogramra, így egyszerre lehet látni a felismert hangokat és a spektrogramot. A hangokat lehet módosítani: odébb helyezni, növelni, illetve csökkenteni a hosszukat, új hangot hozzáadni, korábbit eltüntetni, és a hang erősséget változtatni. Le is lehet játszani az eredeti zenét, illetve a felismert MIDI fájlt, zongorahanggal. Miután a felhasználó elégedett a hangokkal, lehet a zenét MIDI fájlként exportálni.

A programot egy Godot nevű játékmotorban írtam, mert akkoriban azt ismertem a legjobban, illetve könnyen lehet vele valósidejű programokat készíteni, megjeleníteni, és felhasználói felület készítésére is alkalmas. Az itt említett technológiákhoz, és könyvtárakhoz találhatók linkek a megadott, projekthez tartozó GitHub repository-ban¹⁰. A Godot-nak van egy saját szkriptnyelve, a GDScript, ami hasonlít a Python-hoz, és könnyen lehet vele játéklogikát, illetve ezen esetben a program logikáját írni. Ehhez a projekthez viszont nem volt elegendő az elérhető eszközökészlete. Futásidőben még be lehetett tölteni hangfájlt (bár ehhez is egy harmadik fél által készített kis könyvtárra volt szükség), viszont a beépített gyors Fourier-transzformációs (FFT) lehetőség nem volt kielégítő az én céломra, hogy egy pontos spektrogramot ki tudjak rajzolni. A játékmotor viszont ilyen szempontból is könnyen bővíthető, és lehetséges C++ kóddal is kiegészíteni a logikát, a motor újrafordítása nélkül. Így egy C++ könyvtárat használtam a FFT kiszámolására. Sőt, a spektrogram alapján a hangok felismerését végző kódrészletet is C++-ra írtam át végül, és így sokkal gyorsabbnak bizonyult a felismerés.

Maga a felismerés módszere nagyon egyszerű, ami akkoriban logikusnak tűnt számomra. Ahol a spektrogramban nagy az adott frekvencia erőssége, ott tippel egy hangot. Persze azért van egy pár finomítás benne: megnézi a szomszédos félhangokat is, és ha ott is nagy a hangerő, akkor valószínűsíthetőleg valamilyen dobról van szó, ami sok félhangon át

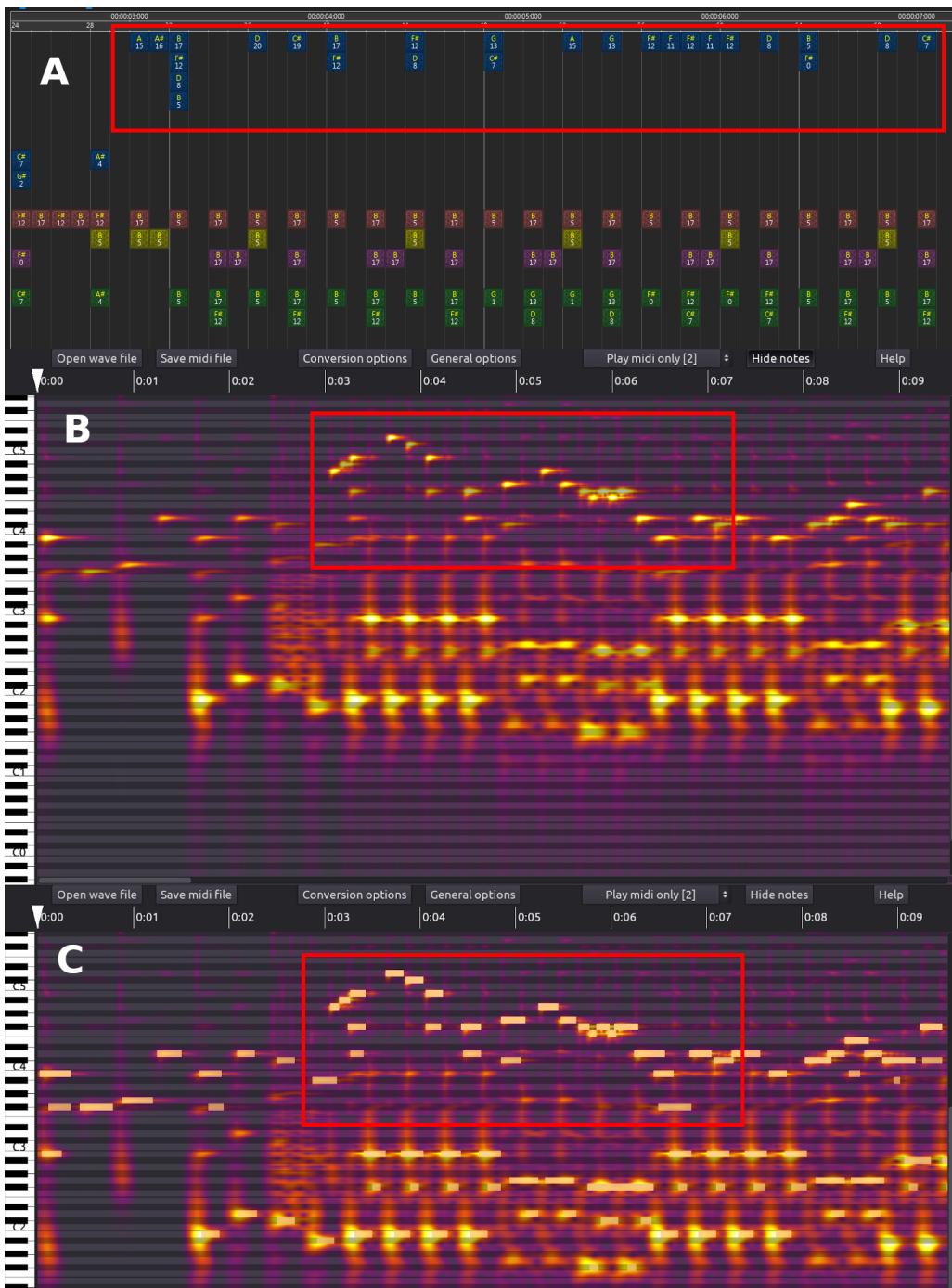
¹⁰<https://github.com/4321ba/MeloMIDI>

terjed, így ezt is figyelembe veszi az adott hang erősségenek számításakor (negatív előjellel). Továbbá egy adott beállítás alapján az egy oktávval mélyebb hangot is megnézi, és kompenzációs céllal kivonja az adott hang erősségréből, így kompenzálva valamelyest a felhangokért. Az egyszerűség ellenére szerintem egészen jól fel tud ismerni zenéket, persze sok szempontból lehetne fejleszteni rajta: ez a felismerés így nem tesz különbséget a hangszerök között, és nem ismer fel ritmust, hogy pár hiányosságát kiemeljem. A GitHub oldalon találhatóak példa képernyőképek és egy bemutató videó, többek között. Sokszor használtam Note Block-os zenékkal is, és a spektrogram nagyon hasznosnak bizonyult, bár a felismert MIDI fájlt nem igazán használtam. Feltehetően nagyon sok program van, ami hasonló jellegű spektrogramokat tud generálni, de én nem találtam olyat, ami pont ilyen célra megfelelőt generálna: Egyszer mellé van rajzolva egy zongora, hogy segítse a hangmagasság kitalálását, másrészt valamilyen módon megoldja, hogy az oktáv távolságok legyenek egyenlő közökre, és ne a frekvenciák. Ezenkívül könnyen lehet belenagyítani a képbe, és odébbtekerni, úgy, hogy a zongorabilentyűk mindenkor a megfelelő hang mellett maradjanak. Az AnthemScore volt a legközelebb, de az egyszerűszt fizetős, másrészt a mélyebb hangokat nem mutatta annyira pontosan.

Ezek kiküszöbölésére saját megoldásokat használtam, amik egyszerűt a célnak jól megfeleltek, és a várt eredményt adták, másrészt utánaolvasással feltehetően lehetett volna matematikailag korrektebb módszereket is használni. Az FFT eredménye egyenlő frekvenciáközönként adott. A cél viszont az, hogy például a 220Hz, 440Hz, 880Hz, 1760Hz stb. zenei A hangok legyenek egyenlő távolságra a képen (hogyha mellé kerül egy vizualizációs célt szolgáló zongora billentyűzete, akkor a billentyűk az elvárt módon, egyenlő méretűek legyenek). Így a választott megoldás az lett, hogy az FFT eredményét interpoláltam mély hangok esetén (ahol kevesebb, mint egy érték tartozott egy pixelhez), és átlagoltam magasabb hangok esetén (ahol több, mint egy érték tartozott egy pixelhez), hogy a pontos frekvenciaértékhez tartozó amplitúdót megbecsüljem a fix frekvenciánkénti adatból. Persze ennél picit kifinomultabb az algoritmus, de a lényege ez.

A másik probléma az volt, hogy nagy FFT méret esetén az időbeli felbontás nem túl jó (ami főleg a magas hangoknál lenne hasznos), kis FFT méret esetén pedig az alacsony frekvenciák esetén nem túl pontos a hangmagasság, így egy-egy hangnál nehéz megmondani, hogy melyik zenei hanghoz tartozik. A megoldás pedig az lett, hogy két spektrogramot számoltam, és összeépítettem őket egy képpé. Az alacsony hangoknál a nagyobb FFT méret dominál, hogy a hangmagasságot könnyű legyen eltalálni, míg a magas hangoknál a kisebb

FFT méretű spektrogramot alkalmaztam, mivel ott már úgyis elegendő a frekvencia-beli felbontás, és így az időbeli felbontás pontosabb lesz. A kettő között pedig interpoláció történt, hogy ne legyen éles a váltás. A lentebbi képernyőképek ezeknek a módszereknek a felhasználásával készültek. Az algoritmus pontos működése megérthető a kódból, GPL (General Public License) alatt elérhető, így bárki megnézheti, és módosíthatja.



Ábra 3: A MeloMIDI program által adott MIDI-ről

A: az eredeti NBS fájl OpenNBS-sel megnyitva

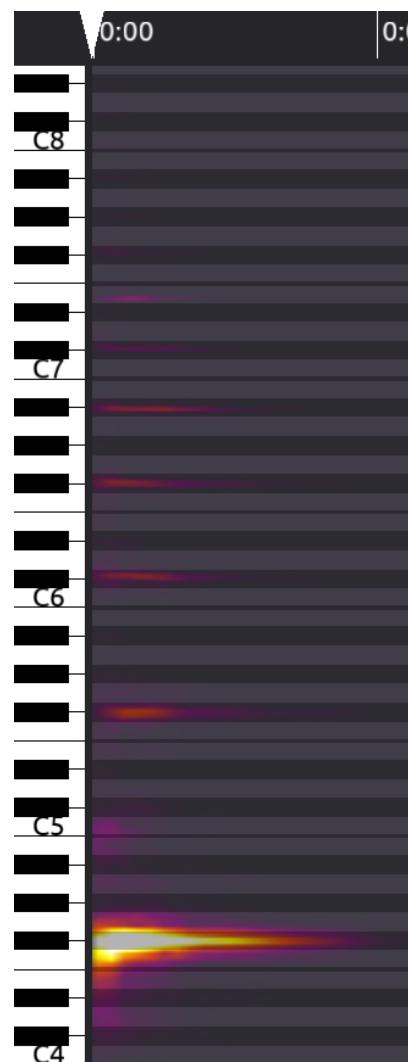
B, C: spektrogram képe felismert hangok nélkül, illetve hangokkal

Példaként itt is megnyitottam ugyanazt a hullámformátumú fájlt, amit fentebb használtam. A 3-as ábra B részén látható a spektrogram, amit a program generált, és ezt érdemes közelebbről összevetni az A jelű részen található eredeti fájllal. A spektrogramról nagyon jól kivehetőek a különböző hangok, illetve az a speciális tulajdonság is, hogy a hangoknak pontosan megegyezik a lecsengésük ugyanazon hangszer esetén. Itt a basszusgitár (mély), és a hárfa (magas) szerepelnek. A basszusgitár eléggé elmosódik, ez sajnos az FFT következménye, és a fentebb írtak alapján végzett javítás ellenére azért a mélyebb hangok nehezebben kivehetőek. A hárfa viszont nagyon szépen látszik felül, és egyértelműen kirajzolódik minden hang tulajdonképpen.

Picit játszva a felismerés beállításaival megkapható a C része az ábrának, ami egészen jól felismeri a bekeretezett részt: a hárfából egyetlen hang hiányzik, és a basszusgitár hangjait is jórészt megtalálta. De a probléma ugyanaz, mint korábban: nincs megkülönböztetés a hangserek között, nincs érdemi időbeli kvantálás, az ütős hangserekkel nem foglalkozik, és nem használja ki az itt egyébként nagyon jól látható formáját a hangmintának.

Ezt hasznos közelebbről is megvizsgálni: a 4-es ábrán látható az egyik hangminta spektrogramja, és közelebbről összehasonlítva a 3B ábrán a bekeretezett hangokkal, lehet látni, hogy lényegében ugyanaz a spektrogram, csak a normalizálás miatt esetleg gyengébb a színe, illetve el van tolva a megfelelő hangmagasságra. A probléma tehát felfogható egy képfeldolgozási problémaként is: ezt a mintát keresem a zene spektrogramjában. Ez kulcsfontosságú, mert így pontosan fel tudjuk használni a rendelkezésünkre álló hangmintát, az ismert lecsengését, és így a speciális formátum speciális tulajdonságait.

Még érdekes megnézni a felhangokat is a képen: többek között ez jellemző a hangszerre. A fő frekvencia F#-nél van (370Hz¹¹) (így van hangolva az összes Note Block-os hangszer), ezután F# (740Hz), C# (1109Hz), F# (1480Hz),



Ábra 4: A hárfa hangminta spektrogramja MeloMIDI-vel

¹¹Frekvenciák: <https://www.staganddagger.co.uk/wp-content/uploads/2022/08/frequency-of-music.jpg>

A# (1865Hz), C# (2218Hz) és E-nél (2637Hz) egy picit lejjebb (Pontosan $7 \times 370 = 2590$). Itt jól lehet látni, hogy a felhangok az alapfrekvencia n-szeresei, ahol n pozitív egész szám. Feltehetően a kiegyenlített hangolás okozza a zenehangok frekvenciái és az alapfrekvencia n-szerese közti apró eltérést, ahol a ténylegesen szóló felhang frekvenciája az n-szeres. Itt lehet jól látni a különbséget a logaritmikus és a lineáris frekvenciaskála között, ugyanis a lineáris skálán egyenlő távolságra lennének ezek a felhangok egymástól, mivel az összes különbsége 370Hz. Az ábrán viszont a zongorabillentyűkhöz tartozó frekvenciák vannak a megfelelő függőleges pozícióban.

2.2.5 Noteblock Music Yoinker

Ez is egy saját projektem volt, és a témába illik. Szintén megtalálható¹² GitHub-on. A célja ugyanaz, mint ennek a projektnek, de a módszere más. Ez egy Java-ban írt Minecraft módosításból áll, ami a Forge nevű módosításbetöltő keretrendszert használja, illetve Python nyelven írt feldolgozó szkriptekből.

A Minecraft modolás, módosítások írása is népszerű, apró változtatásoktól teljesen új játékmechanikákat is hozzá lehet adni a játékhoz, és bár hivatalosan nincs támogatva, a fejlesztők is tudják, hogy a játéknak ez is hozzájárul a sikéréhez. Így bizonyos módokon elősegítik a kód visszafejtését, például kiadják az obfuscaciós leképezést¹³, az osztály- és függvénynevek visszafejtéséhez.

A Noteblock Music Yoinker Forge mod része a kliensen betölthető módosítás, és a játék által kiadott Note Block hangokat naplózza egy CSV (Comma-Separated Values) fájlba, ezzel megszerezve a ténylegesen szóló hangok listáját.

A Python szkriptek pedig sokféle transzformációt képesek elvégezni. Egyik fontosabbik a több felvétel kiátlagolása, illetve összehasonlítása, a pontatlan időzítés (lag) miatt szükség van általában arra, hogy több felvétel alapján lehessen kitalálni a szándékolt időzítést. Van még MIDI-, illetve NBS-exportáló szkript is, többek között.

A szkriptek több szempontból is hasznosak ezen projekt szempontjából, egyrészt már személyesen jobban ismerem az NBS fájlformátumot, másrészt pedig létezik egy szöveges (CSV alapú) fájlformátum, amely képes leírni a Note Block-os zenéket, és így könnyebb

¹²https://github.com/4321ba/noteblock_music_yoinker

¹³https://minecraft.fandom.com/wiki/Tutorials/See_Minecraft%27s_code

összehasonlítást tesz lehetővé. Ez akkor lesz szükséges, ha a konverzió minőségét szeretném mérni.

Ennek segítségével a WynnCraft nevű Minecraft szerver zenéiből elkészítettem az eredetihez közel álló NBS fájlokat, ezen adathalmaz hasznos lesz a pontosság mérésében. Ez több, mint 200 zenéből áll (175 eredetileg, de van, aminek több verziója, vagy része található meg), és a YouTube-ra Creative Commons licenc alatt került feltöltésre¹⁴, így én is közzétehettem a felismert fájlokat¹⁵. Ez egy óriási projekt volt, de végül az akkor elérhető összes zenét felvettek, és feltöltötték YouTube-ra is, vizualizációként, zongora oktatóanyag címszó alatt¹⁶, a Creative Commons licenc ezen verziója megengedi az újrafelhasználást. Fontos kérdés viszont ilyen tematikájú programok esetében a szerzői jog figyelembe vétele. A program önmagában nem sért szerzői jogot, azonban hogyha valaki olyan zenén futtatja, aminek a szerző nem engedte meg az újrafelhasználását, majd közzéteszi, vagy felhasználja a publikus Minecraft szerverén, azzal biztosan szerzői jogot sért.

2.2.6 Galaxy Jukebox

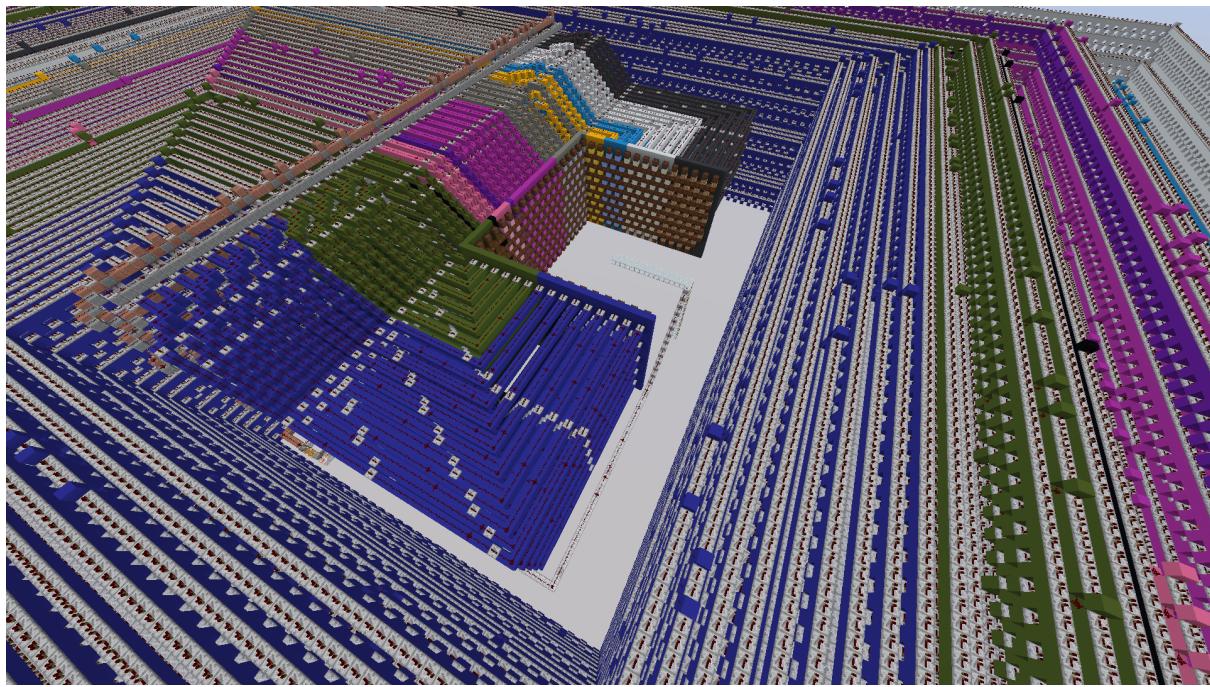
Ez is egy saját projekt¹⁷, NBS fájlból képes olyan Minecraft-beli vöröskő áramkört létrehozni, ami az adott zenét lejátssza, a játékon belül. Sok szempontból javít a Note Block Studio-ba épített exportőrön, illetve más logikával működik. Ez is egy parancssori alkalmazás (bár tartozik hozzá egy egyszerű felhasználói felület is), aminek a bemenete az NBS, és a Minecraft-ba beilleszthető világdarabot generál (schematic fájl). Egy példa az 5-ös ábrán látható.

¹⁴<https://www.youtube.com/watch?v=AaKzpBH65FE&list=PLyqkjDCr8kbI3CjNZimiri8shU1GbfJ6E>

¹⁵https://github.com/4321ba/WynnCraft_Noteblock_OST

¹⁶https://www.youtube.com/playlist?list=PLDa4Vj43E2e-mCukFLGM5xTLILNZ_daaI

¹⁷https://github.com/4321ba/Galaxy_Jukebox



Ábra 5: *Galaxy Jukebox*

2.2.7 NBSWave

Az Open Note Block Studio-nak (1.4) az új NBS-ből hullámformátumú hangfájl szintetizáló alprogramja. MIT licenc alatt elérhető a GitHub-on¹⁸, és a projekt szempontjából azért bír különös jelentőséggel, mert az ellenkező irányú konverziót valósítja meg. A régi exportörnek többféle problémája volt, így az általa generált zenék pontos felismerése nem elsődleges célja ezen szakdolgozatnak.

Ezen kívül Note Block-os zenét többféle módszerrel is lehet szintetizálni, például: felvenni videófelvezővel az ONBS visszajátszását, a Minecraft-ba exportált vöröskő-szerkezet hangját, vagy a Minecraft-ba exportált, parancs, vagy plugin (szerver oldali mod) alapú lejátszó hangját. Ezek különböző szempontokból lehetnek pontatlannak: időzítés, túlvezérlés, illetve hangmagasság apró eltérése szempontjából. A legtisztább eredményt az nbswave használata adja, azt fogom a pontosság ellenőrzéséhez használni.

2.3 Választott módszer, technológia

2.3.1 Felismerés módszere

Láthattuk, hogy általános, hullámformátumú fájlt MIDI-vé konvertáló megoldás sok létezik, amelyek adnak egy eredményt, viszont több szempontból nem túlzottan kielégítőek.

¹⁸<https://github.com/OpenNBS/nbswave>

Ezek jórészt neurális háló alapú megoldások. Az a feltételezésem, hogyha a fentebb említett speciális tulajdonságokat kihasználjuk, ennél jobb felismerés lesz lehetséges.

A [19]-ben leírtaknak megfelelően a szokásos folyamatot alkalmazom: a hullámformátumú hangfájlt, a bemenetet, átszámoltatom egy idő-frekvencia reprezentációjával (a gyors Fourier-transzformációt, FFT-t használva), majd ebből számolható a kimenet, a szóló hangok időpillanatonként.

A spektrogram alapján történő felismeréshez pedig a képek összehasonlítását használom fel (lásd 2.2.4), két spektrogram között definiálok egy hibafüggvényt, amit minimalizálni próbálok az adott időpillanatbeli hangok különböző kombinációjával. Ezzel a módszerrel az időzítést is pontosan lehet tudni, és kihasználhatók az ismert hangminták, és a lecsengés is. Eredményképpen pedig pontosan fogom tudni, hogy mely hangminták milyen hangerővel kellett, hogy szóljanak, hogy kiadják az eredeti spektrogramot.

Ez hasonlít a [27]-ben leírtakhoz: ott is minimumkereséssel keresik a mátrix szorzótényezőit. Viszont azért sokban különbözik is: a dimenziók nem egyeznek meg, a cikkben az egész spektrogramhoz egy minimumkeresés tartozik, az előző bekezdésben leírt módszer ellenben egy adott időpillanatot vizsgál csak, és így pontosan ki lehet használni, hogy ott kezdenek szólni a hangok. Továbbá a minimumkeresésnél fontos kérdés lesz a deriváltfüggvény meghatározhatósága, hogy lehet-e ezt felhasználó minimalizáló algoritmust választani.

2.3.2 Választott technológia

Nagyon sokféle programozási nyelv, és fejlesztési környezet lenne erre a problémára megfelelő. Ami szükséges, hogy legyenek megfelelő könyvtárak a fájlformátumok olvasásához, illetve a szükséges számításokhoz: az FFT-hez, az újramintavételezéshez, és a minimumkereséshez. Fontos még, hogy platformfüggetlen legyen. Ezeknek többek között a Python és a Rust is megfelelne. Előnyös, ha minél több problémát minél korábban jelez a fordító, illetve futási sebesség szempontjából is jobb a Rust, így arra esett a választásom.

3 Tervezés

3.1 Technológiák ismertetése

3.1.1 Note Block-os zenék sajátosságai, formátuma

A Note Block-os zenéknek, mint az fentebb többször meg volt említve, több sajátossága van, ami segíti a felismerést az általános esethez képest. Most ezeket fejtem ki jobban. Ezek jórészt a Minecraft limitáltságából adódnak. Érdekes ugyanakkor, hogy milyen szép zenéket lehet a szűk lehetőségek ellenére is alkotni.

16 féle hangszer van, de korábbi Minecraft verziókban sokáig csak 5-6 volt. Emiatt az egyik elérhető adathalmaz, a WynnCraft nevű szerver korábbi zenéi csak 5-féle hangszert használnak.

Hangmagasságból a játékon belül 25 elérhető (2 oktáv, kiegyenlített hangolással¹⁹). Az ONBS szerkesztőprogram megengedi ezen kívüli hangok felvételét is, viszont pirossal jelzi az inkompatibilitást. A felismerő programban ezt úgy lehetne lekövetni, hogy parancssori opcióként a felhasználó képes legyen a 2 oktávon kívüli hangok felismerésére is, viszont alapértelmezetten csak 2 oktávval próbálkozzon a program.

Hangerő: 0.0 és 1.0 között célszerű reprezentálni, a szerkesztőprogram 0% és 100% között jelzi. Problémás a felismerése, amennyiben a túlvezérlés elkerülése érdekében az nbs to mp3 exportőr, vagy csak simán a Studio maga, lejjebb veszi a hangerőt. Akkor minden más felismerés is problémás, ha nem veszi lejjebb a hangerőt, és ezért sok helyen túlvezérlés van. 1.0 fölé is mehet a hangerő, ha egyszerre ugyanaz a hangszer ugyanazon a hangerőn többször is szól, ez elég gyakori a WynnCraft-os adathalmazban. Ilyen lehetőség a MIDI fájlformátum, és a formátummal dolgozó programok esetében túlnyomórészt nem támogatott.

Időzítés: a tempót tps-ben mérjük, ami a tick per second rövidítése. Ez ugyanaz a dimenzió, mint a bpm (beat per minute), csak nem per perc, hanem per másodperc, és nem a negyedeket számoljuk, hanem a zenében található legkisebb időbeli különbséget két szóló hang között (=tick), ami általában a 16-odoknak felel meg. A szokásos értéke 5, 10 és 20, előfordul még 6.75 is, 6.67 helyett. Régebben 0.25 többszörösére kerekítette a Studio a

¹⁹https://en.wikipedia.org/wiki/12_equal_temperament

tempót, ma már bármilyen előfordulhat, bár gyakorlatban még mindig gyakran a .25-ös kerekítést használják, és az elég finom is. A játékban a fentebbi 4 játszható le tökéletesen, de a Studio-ban bármilyen tempóval lehet írni zenét. A tempó a zene során állandó, azt nem lehet megváltoztatni, viszont a felvételben előfordulhat időbeli pontatlanság (akár az exportőr pontatlansága miatt). Ezt szükséges dinamikusan érzékelni, mert amennyiben egyenlő időközönként feltételezzük az ütemeket, és ennek a valóságalapját nem ellenőrizzük, könnyű egy 2 perces zenénél a végére elcsúszni egy tizedmásodpercet például, és azzal már elromlik a felismerés, amennyiben azt feltételezi, hogy az ütem elején indulnak a hangok.

A kimenő fájlformátum a Note Block Studio saját fájlformátuma, az nbs. Leegyszerűsítve egy 2D Note Block-ok tömbjét kell kiadnunk. vízszintes tengelye az időt reprezentálja, egy-egy oszlopot tick-nek nevezünk. Egy-egy sor neve a layer, és általában egy hangszer szoktak csak rátenni. A 6-os ábrán a Note Block Studio látható, ő különböző színekkel jelzi a különböző hangszereket.



Ábra 6: Képernyőkép az ONBS-ről, a Note Block-ok megjelenése a programban

3.1.2 Spektrogram

A spektrogram nagyon fontos szerepet tölt be a felismerésben. Vízszintes tengelye az időt reprezentálja, függőleges tengelye pedig a frekvenciát. Példa található a 3-as, és a 4-es ábrán. A képet különböző színekkel lehet színezni, de mindenhol az adott x-y koordináta, azaz az adott időpillanatban szóló hang adott frekvenciájának erősségét jelzi a szín.

Ennek generálása, amint azt a [25] cikkben is részletezték, úgy történik, hogy először rövid időközöket kivágunk a hullámformátumú fájlból, például 4096 hosszú mintákat. A 4096 egészsges középút az időbeli pontosság, és a mély hangok nagyobb felbontása között. Ezután egy ablakfüggvényt alkalmazunk rá, hogy a vágásoknál található hirtelen ugrás a lehető legkevésbé zavarjon bele a frekvenciákba, olyan frekvenciák megjelenésével, amik az eredeti

zenében nem voltak. Erre kell az FFT-t alkalmazni, ami így még apróbb konverziók után (komplex számok abszolút értéke) kiad egy oszlopnyi pixelt. Ugyanezt elvégezve, például 1024 mintával jobbra, ki lehet adni a spektrogram pixeleinek következő oszlopát. Az, hogy az FFT bemenetének mérete kettőhatvány, a hatékonysságon segít.

Ezután a felhasználási célnak megfelelően, ahogy a 2.2.4 fejezetben részletesebben kifejtésre került, célszerű lehet áttérni lineáris frekvenciaskáláról logaritmikusra, mivel az felel meg jobban a füllel észlelhető hangok közti különbségeknek. Ezen projekt céljából feltehetően nem lesz szükség az áttérésre.

3.1.3 Újramintavételezés

A hullámformátumú hangfájl nyers formában (tömörítetlenül) mintákból áll (másodpercenként 44100-ból például), és egy mintavételezési rátából (sampling rate), ami egy egész, 44100 például. Hogyha a hangszóró valamilyen oknál fogva 48000Hz-cel tud hangot kiadni, felmerülhet a kérdés, hogy a 44100Hz-es mintavételezésű hangfájlból hogyan készítünk 48000Hz-es mintavételezésűt.

Erre vannak algoritmusok, amik valamilyen módon megbecsülik, hogy a folytonos hanghullám a másodpercenkénti 44100 minta alapján eredetileg milyen formát vehetett fel. Ennek segítségével újramintavételezi a hangfelvételt, azaz meghatározza a másodpercenkénti 48000 minta értékét. Így lejátszva ugyanazt a zenét halljuk, mint korábban, csak a belső reprezentáció változott meg.

Számunkra ugyanez az algoritmus szükséges, csak kifordítva. A Note Block-os minták elérhetőek, viszont egyetlen hangfájl tartozik egy-egy hangszerhez. Ennek segítségével kell kiszámolnunk az elérhető 2 oktávnyi potenciálisan megszólaló hangot. Azt lehet tudni, hogy a játék egyszerűen torzítja a hangot, tehát az egy oktávval mélyebb hang kétszer annyi ideig szól, mint az eredeti. A hullámforma ugyanaz, csak rövidíteni- illetve nyújtani kell. Ennek pedig az újramintavételezés a technikája: amennyiben egy oktávval mélyebb hang előállítása a cél, újra kell mintavételezni a hangot 44100Hz helyett 88200Hz-cel, ezzel kétszer annyi minta lesz ugyanannyi időre, majd 44100-as mintavételi frekvenciájú hangfelvételként kell értelmezni, így kétszeresére nyúlik az idő, és a mintavételezési frekvencia marad az eredeti, a hang pedig egy oktávval mélyebb lesz.

Ugyanígy könnyen számolható a többi hang is, a kiegyenlített hangolás [29] segítségével. Ez azt jelenti, hogy egy oktávot, azaz a kétszeres frekvenciasorozt felosztjuk

egyenlően 12 részre, félhangonként. Mivel logaritmikus a skála, így az egyenlő felosztás n-nel való osztás helyett n-edik gyökvonást jelent, tehát egy félhang frekvenciája $^{12}\sqrt{2}$ -szeres frekvenciaszorzóval számolható. Így vegyük észre, hogy 12 félhang pont kiadja a 2-szeres frekvenciaszorzót az oktávhöz, illetve például a 7 félhang majdnem kiadja a tiszta kvintet ($2^{7/12} \approx 1,498 \approx 3/2$). A többi tiszta hangköz is elegendően közel van a $2^{1/12}$ hatványaihoz. Az óriási előnye pedig a tiszta hangoláshoz képest, hogy minden hangnemben ugyanannyira hamis.

Ennek megfelelően az újramintavételezéshez a mintavételi frekvenciát $2^{-(p-12)/12}$ -szeresére kell változtatni, ahol p 0-tól 24-ig terjed, és 12 felel meg az eredeti minta hangmagasságának, az 1-szeres szorzónak. A negatív előjel (vagyis osztás) azért szükséges, mert 44100Hz-et 2-vel osztva, majd a mintákat újraértelmezve 44100Hz-en, kapjuk meg a kétszeres frekvenciájú, azaz fele olyan hosszú hangmintát.

3.1.4 Optimalizációs algoritmusok

Ha adott egy költségfüggvény, annak minimalizálására sokféle algoritmus létezik. Ennek a projektnek a keretében a Nelder-Mead, a Steepest Descent és a Nonlinear Conjugate Gradient módszereket tesztem, amint az 5.2.2 fejezetben részletesebben kifejtem. Az előző féléves dokumentációban bemutattam a Nelder-Mead módszert²⁰, és a másik kettő ezen félévben tesztelt algoritmus (mivel a gradienst is felhasználja) sokkal jobb eredményt adott, így a Nelder-Mead módszert külön nem részletezem.

A Steepest Descent módszer [30] egy minimumkereső eljárás, ami egy n-dimenziós térben keresi egy költségfüggvény minimumát. Ehhez felhasználja a gradienst, viszont másodrendű deriváltakra nincs szükség. Az algoritmus a következő: elindulunk egy kezdeti pontból, meghatározzuk a legnagyobb lejtő irányát, és arra lépünk egy picit. A lépés mértékét egy vonal menti keresés határozza meg, ami megbecsüli, hogy meddig lehet lépni, hogy a legnagyobb csökkenés legyen elérhető a költségfüggvényben.

A Nonlinear Conjugate Gradient módszer szintén n-dimenziós térben keresi egy költségfüggvény minimumát, a gradiens felhasználásával. Az algoritmusa nagyon hasonló a Steepest Descent-éhez, de a lépés mértékének kiszámolásához a korábbi gradienst is felhasználja [30].

²⁰<https://github.com/4321ba/mp3-to-nbs/blob/main/documentation/paper/documentation.md#optimaliz%C3%A1l%C3%A1s>

3.2 Követelmények

3.2.1 Funkcionális

Az alkalmazás futtatható legyen parancssorból, és meg lehessen adni egy bemeneti fájlnevet, és egy kimeneti fájlnevet. Helytelen használat esetén, ezenkívül a megfelelő parancssori kapcsolóval (--help) írja ki a használati útmutatót, az elérhető kapcsolókat, illetve azt, hogy hogyan kell megadni a fájlneveket, és azok mit jelentenek.

Az alkalmazás megfelelő bemenet esetén a kimeneti NBS fájlba írja be a bemeneti hullámformátumú fájl NBS formátummal közelített reprezentációját, a szakdolgozat többi részében taglalt módszer alapján a lehető legjobb közelítést elérve.

3.2.2 Nem funkcionális

Nem funkcionális követelmények közül a konverzió sebessége, és a felismerés pontossága a legfontosabbak. A sebesség a kutatásunk során másodlagos szempont a felismerés pontosságához képest, így amennyiben lehetséges, a felismerés pontosságának növelése a cél a sebesség rovására, meghatározott kereteken belül. A valósайдejűség nem elvárás, és a sebesség nagyon fog függni a felhasznált számítógéptől. A pontosság számértékkal való mérése az 5.2.1 fejezetben van részletezve.

Fontos még kitérni a robusztusságra, ugyanis a hullámformátumú fájlok sok szempontból robusztus kezelést igényelnek. Ennek részletei a 2.2.7 fejezetben találhatóak. Az elsődleges cél az nbswave által generált, nem túlvezérelt, és időben is pontos hangfájlok felismerése.

Megemlíthető a szabványkompatibilitás még, mint nem funkcionális követelmény. A bemenő fájlt egy könyvtár fogja kezelní, de szabványos wav, mp3 és ogg fájlok beolvasása mindenkorban lehetséges a programmal. A kimenő fájlformátum az új, ONBS által használt [11] NBS formátum 4-es verziója, mivel ez támogatja a Note Block-szintű hangerőt. Korábbi verzióval csak layer-enként lehetett hangerőt állítani.

3.3 Rust könyvtárak bemutatása

Fontos a megfelelő könyvtárak megtalálása, és felhasználása, hogy ne implementálunk feleslegesen olyan logikát, ami már készen elérhető egy könyvtárral. A Rust ökoszisztemája könnyű és egyértelmű módszereket biztosít függőségek leírására, és azok

automatikus letöltésére. A könyvtárak túlnyomó része elérhető a közösség hivatalos könyvtárnyilvántartásában, a crates.io-n. Az alábbiakban bemutatom, hogy melyik feladatra milyen könyvtárat találtam. Mindegyik könyvtár a MIT licenc alatt elérhető (Massachusetts Institute of Technology által kiadott, permisszív licenc²¹), így biztosan nem okoz problémát a felhasználásuk, akkor sem, hogyha a projekt kódját én is MIT licenc alatt teszem közzé.

3.3.1 Babycat

A Babycat²² nevű könyvtár hangdekódolással, és -manipulációval foglalkozik. Nem túlzottan elterjedt (az íráskor 6400 letöltéssel), mivel leginkább több könyvtár összecsomagolása. A hangfájlok dekódolását a Symphonia nevű könyvtár végzi, ami több, mint 1.000.000 letöltéssel rendelkezik. Ami miatt mégis a Babycat-re esett a választásom, az az, hogy a libsamplerate nevű C könyvtárat is biztonságos módon (safe) a felhasználó rendelkezésére bocsátja, ami újramintavételezést tesz lehetővé, és ez számomra feltétlenül szükséges. Wav fájlt készíteni is tud, ami hibakereséshez hasznos tud lenni.

3.3.2 Spectrum analyzer

A spectrum-analyzer²³ egyszerű, könnyen használható (KISS, keep it simple, stupid) FFT-t absztraktáló könyvtár. Továbbá beépítve tud ablakfüggvényt is alkalmazni a mintákra. A készítője írt egy blogposztot is²⁴, ami röviden, tömören, és jól érthetően elmondja, hogy mit, hogyan kell számolni.

3.3.3 NBS fájlformátum-kezelő

Az nbs-rs²⁵ könyvtár segítségével könnyen kezelhetőek az NBS fájlok Rust-ból. Mivel a fájlformátum nem széles körben használt (a wav vagy az mp3 formátumokhoz viszonyítva legalábbis), és a programnyelv sem a legelterjedtebb, így némi kellemes meglepődést okozott, hogy létezik a problémára könnyen használható könyvtár. Példakód is elérhető egy fájl létrehozására, ami tökéletes a projekt céljára.

²¹<https://mit-license.org/>

²²<https://crates.io/crates/babycat>

²³<https://crates.io/crates/spectrum-analyzer>

²⁴<https://phip1611.de/blog/frequency-spectrum-analysis-with-fft-in-rust/>

²⁵<https://crates.io/crates/nbs-rs>

3.3.4 Argmin

Az argmin²⁶ matematikai optimalizációs algoritmusokat tartalmaz tisztán Rust-ban. Nagyon sokféle algoritmus elérhető, konzisztens interfésszel. Vannak olyanok, amikhez szükséges a derivált, például a steepest descent, illetve olyanok is, amikhez nem kell: ilyen a Particle Swarm Optimization, illetve a Nelder-Mead method. Hogyha a megoldáshoz hibát szeretnék minimalizálni, ennek a könyvtárnak a segítségével fel lehet használni egy-egy gyors, hatékony, és sokak által tesztelt algoritmust.

3.3.5 Aubio

Ritmusdetektálásra nem találtam könnyen használható Rust könyvtárat. Az aubio²⁷ az azonos nevű C könyvtár kötése, ami sokféle, komplex funkcióval rendelkezik, események érzékelésére és ritmus meghatározására is használható. Licencelési szempontból egy fokkal problémásabb, mivel GPL (GNU General Public License, copyleft licenc²⁸), így feltételezhetően²⁹ a licenc pontos betartásához a felhasználása esetén a teljes projektet GPL alatt volna szükséges elérhetővé tenni. Ez megnehezíti azon potenciális felhasználók helyzetét, akik MIT licencű projektükbe szeretnék ezen konvertert integrálni. Ilyen valószínűleg nem fog megtörténni, de az ONBS MIT licenc alatt érhető el, emiatt szeretném, ha megmaradna a lehetőség arra, hogy a projektet licencelési problémák nélkül integrálják, vagy mások MIT licenc alatt fejleszthessék tovább. Ennek ellenére, amennyiben könnyűnek bizonyul a használata, és jó eredményt ad, a licencelési problémáktól eltekintek.

3.4 Architektúra

Így, hogy az építőblokkok bemutatásra kerültek, felvázolható az architektúra, azaz mely építőelemek, milyen sorrendben, hogyan fogják elősegíteni a cél elérését.

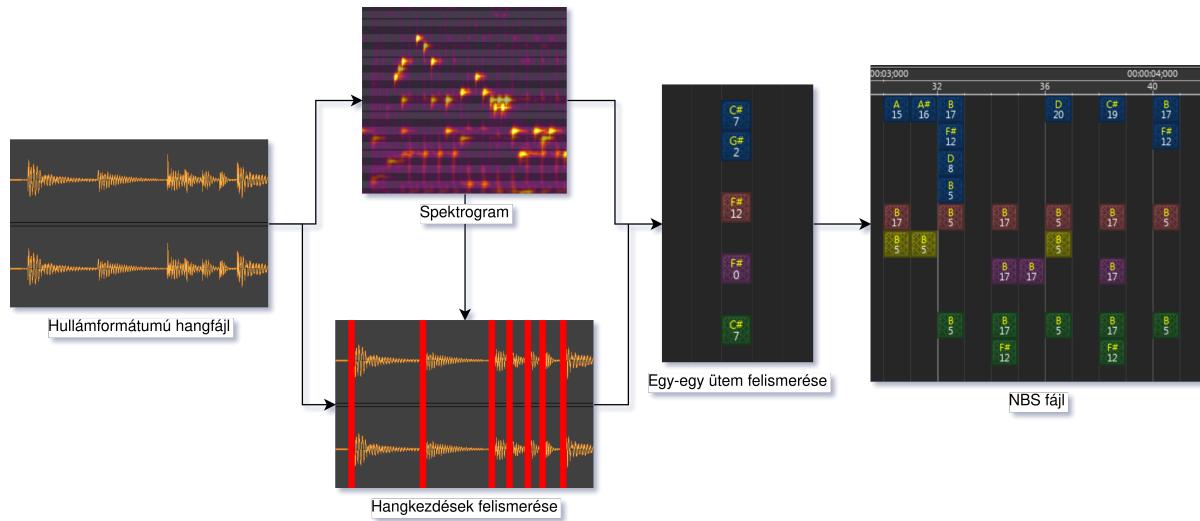
²⁶<https://crates.io/crates/argmin>

²⁷<https://crates.io/crates/aubio>

²⁸<https://www.gnu.org/licenses/gpl-3.0.html>

²⁹<https://opensource.stackexchange.com/questions/1640/if-im-using-a-gpl-3-library-in-my-project-can-i-license-my-project-under-mit-1>

3.4.1 Áttekintés



Ábra 7: Áttekintés az architektúráról

A 7-es ábrán látható egy áttekintés az alkalmazás tervéről. Bemenetként megjelenik a hullámformátumú hangfájl, melyből legenerálható a spektrogram, illetve felismerhetőek a fontos időpillanatok, ahol potenciálisan Note Block-ok kezdőpillanatai vannak. Ezután ezekben az időpillanatokban kielemezve a spektrogramot, minimumkereséssel megkapjuk az abban a tick-ben tippelt Note Block-okat, amit ha az összes időpillanatra elvégzünk, megkapjuk a teljes felismert NBS fájlt.

3.4.2 Egy ütem felismerése

Ami további részletezést kíván, az az egy pillanat felismerése minimumkereséssel. Itt felhasználható a spektrogram adott időpillanatbeli darabja, az összes hangminta, illetve spektrogramjaik, és szükség esetén az előző időpillanatokból átnyúló hangok hullámformája is. Egy minimumkereső algoritmussal, például a Steepest Descent-tel elég lassan, de pontos eredményt lehet elérni a következő módszerrel.

N darab változtatható paraméterünk van, ami a hangminták hangereje lesz, 0 és 1 között. Ez 5 hangszer esetén, hangszerenként 25 hangmagassággal 125 paraméter. Ezek változtatásával keresünk egy olyan kombinációt, hogy ha az összes hang a paraméter által adott hangerején együtt szól, akkor a spektrogramja a lehető legközelebb legyen a felismerendő fájlból származó spektrogramhoz. Ehhez egy hibafüggvényt kell definiálni. A hibafüggvény a következőkből áll: lineárisan kell kombinálni a hangminták spektrogramjait a hangerőkkel (aktuális paraméterértékek), mint súlyok, illetve hozzájuk adni a korábbi

ütemben felismert hangok lecsengését. Figyelni kell arra, hogy a következő ütem kezdete előtt érjen véget a spektrogram, és ennek a távolságát mérjük meg az eredeti spektrogram megfelelő darabjától. Kétféle lehetséges távolságmérték például a spektrogramok pixelenkénti különbségének a négyzeteinek az összege, illetve az abszolútértékeik összege. Az FFT komplex számokat ad eredményül, így az az információ is felhasználható.

Egy másik módszer a hangminták hullámformájának lineáris kombinációja, és annak Fourier-transzformációja, minden költségfüggvény-híváskor. Azt is ugyanúgy össze kell hasonlítani az eredeti spektrogrammal. Ennél a módszernél lehet a túlvezérlést szimulálni a hullámformák összeadásakor, egyéb esetben viszont, a Fourier-transzformáció linearitása miatt ugyanazt az eredményt fogja adni, mint a spektrogramok lineáris kombinációja, azzal a módszerrel viszont kevesebb FFT-t kell számolni.

Az állapottér elég nagy, viszont egy adott időpillanatban a lehetségesen szóló hangoknak csak a töredéke fog ténylegesen szólni. Sokat segíthet a felismerésen, ha végünk egy előfelismerést: melyek azok a hangok, amik biztosan nem szólnak, amiket ki lehet hagyni az optimalizálásból teljesen. Ez úgy történik, hogy a hangminta spektrogramját kivonjuk a felismerendő zenerészlet spektrogramjából (pixelenként), majd egy rámpafüggvényt ($f(x)=\max(0,x)$) alkalmazunk rá, pixelenként (azaz minden adott időpillanat-frekvencia párrosra). Majd a különbségeket összeadjuk, és ez alapján egy határértékkal döntjük el, hogy elfogadjuk-e az adott hang létezésének lehetőségét. Ez a féloldalasság azért szükséges, hogy azt ne szankcionáljuk, ha sok hang van a zenerészletben, azt viszont igen, ha pont az a frekvencia, aminek ott kellene lennie, az nincs ott. Érdemes továbbá az alacsony hangerejű hangok felismerése érdekében csendes mintával összehasonlítni a spektrogramot, hogy azt is felismerje, mint potenciálisan szóló hang.

4 Önálló munka bemutatása

A szoftvert Rust nyelven valósítom meg, annak 1.82-es verzióját használom. A forráskód verziókövetéséhez GitHub-ot használok, így a forráskód itt³⁰ érhető el, licencnek a 3.3.5 fejezetben taglalt érvek miatt a MIT licencet választottam. Ez a projekt volt a téma az előző (2023/24/2) féléves Önálló laboratórium tárgyamnak is. Annak eredményeképpen két zenén tesztelt kezdetleges konverzió már felismerhető eredményt adott. Ebben a félévben nagy mennyiségű adaton is mérhető szoftver készítése, illetve a mérhető eredmény javítása szempontjából történt jelentős előrehaladás. Konkrét fejlesztések közé tartozik az aubio könyvtár felhasználása ritmusfelismerésre, a költségfüggvény módosítása, hogy az FFT komplex eredményének fázisát is felhasználja a felismerés pontosításához, illetve a költségfüggvény deriváltjának meghatározása, hogy azt felhasználó optimalizáció is felhasználható legyen, ezzel nagyon sokat pontosítva a felismerésen. A program nagy adathalmazon való tesztelése, az ezt lehetővé tevő szkriptek létrehozása, a mérések elemzése, illetve ezen dokumentum megírása is ebben a félévben történt.

4.1 A Rust nyelv

A Rust³¹ sok szempontból érdekes nyelv. Leginkább a C és C++ közé tudnám besorolni a tanult nyelvek közül. Modern, azaz sok, új programnyelvekben található egyszerűsítés, szintaktikai cukorka található benne. Ezenkívül viszont sok szempontból eltér egy „szokásos” modern, magas szintű nyelvtől: nincs benne szemétygyűjtő (garbage collector), ennek ellenére garantálja a biztonságos memóriaeléréseket a C/C++-szal szemben, a memória fordítási idejű nyomonkövetésével (borrow checker).

Továbbá lehet tagfüggvényeket definiálni egy saját adattípushoz (struct, enum), viszont nincs benne öröklés, ami sokat egyszerűsít a nyelven. Kivételek helyett C-szerű megoldást használ, azaz a visszatérési érték része a hiba is, viszont kikényszeríti a hiba explicit ellenőrzését, ami véleményem szerint biztonságosabb, könnyebben követhető, és jobban látható, mint a kivételek használata. Emellett szintén sok sarokeset ismeretének szükségességtől megkíméli a programozót: mi történik például, ha szülőre mutató referencián keresztül hívunk leszármazott osztályban felüldefiniált függvényt, ami visszahívja

³⁰<https://github.com/4321ba/mp3-to-nbs>

³¹<https://www.rust-lang.org/learn>

a szülő implementációját, és az kivételt dob. Persze sok nyelven létezik megoldás a kivételdobás lehetőségének jelzésére, de szerintem könnyebben átlátható, ha minél kevesebb nyelvi szintű funkciót vezet be a nyelv, és a már meglévő, könnyen érhető rendszert (itt a visszatérési értéket, és kényelmesen használható tagfüggvényeket) használjuk a hibák jelzésére is. Ide tartozik a null-biztonság is: egy változó nem lehet null, hacsak explicit ez nincsen jelezve, ekkor viszont minden használatkor le kell kezelni a null (None) esetet is.

A fordító a lehető legtöbb hibát kiszűri fordítási időben, így többek között pontos típusinformációknak is rendelkezésre kell állniuk. Ezt viszont képes a változók használatából is kikövetkeztetni. A megváltoztathatatlan (immutable) változók, illetve referenciai az alapértelmezettek, de lehet megváltoztathatót is létrehozni, amennyiben szükséges. Ez segít átgondolni, hogy tényleg szeretnénk-e, hogy egy függvény megváltoztathasson egy adott változót, és amennyiben ez explicit nincs megengedve, a fordító szólni fog.

4.2 Modulok bemutatása

A következő alfejezetekben részletesen bemutatom a forráskódot, a logikáját, a megírt függvényeket, és a velük kapcsolatos döntéseket. Modulonként haladok. Egy-egy modul ebben a projektben egy-egy Rust fájt jelent. Bottom-up sorrendet használok, azaz a végére hagyom a fő logikát implementáló modulokat, hogy minden a már bemutatott modulokra kelljen csak hivatkoznom.

4.2.1 Complex spectrum, Observer

Ez a két modul máshonnan átvett kód, de a projekt része, mivel így lehetséges felhasználni. A spectrum-analyzer crate nem teszi elérhetővé az FFT komplex eredményét, csak annak amplitúdóját, nekem pedig szükségem volt a komplex eredményre is. A megoldás az FFT könyvtár ([microfft³²](https://crates.io/crates/microfft)) közvetlen használata volt, viszont a spectrum-analyzer könyvtár egyéb kényelmi funkciókkal is ellátta a felhasználót: FFT-méret beépítése a függvény nevébe makróval, illetve a DC komponens kicsomagolása ami számomra feltűnt. Emiatt úgy döntöttem, a spectrum-analyzer könyvtár kódját használom fel, és módosítom úgy, hogy komplex számok vektorát adja vissza az amplitúdók helyett. Ezt megtehetem, mivel a spectrum-analyzer is MIT licenc alatt érhető el. Ennek a kódja található a complex-spectrum modulban, ami a `samples_fft_to_complex_spectrum` függvény hívásával elérhető.

³²<https://crates.io/crates/microfft>

Az Observer modul szolgál a tracing nevű naplózó könyvtár, és az argmin nevű optimalizáló könyvtár összekötésére, azaz arra, hogy az optimalizáló algoritmus státuszjelentését a tracing könyvtáron keresztül írja ki a kijelzőre, és így konzisztens legyen a többi kiírással, és könnyen ki-be kapcsolható legyen. Ezt a módosítást egy RedstoneWizard08 nevű GitHub felhasználó küldte, és a tracing crate használatát is ő vezette be, egy pull request keretében³³. A húzási kérelmet végül nem fogadtam el, mivel jelentős takarítást is végzett, amik egy részével nem értettem egyet, illetve másrészről pedig én bátrabban törlök ki nem használt kódrészleteket, amiket ő csak inkább kikommentezett, hogy lehetőleg ne zavarjon be a projektbe. Viszont így is nagyon örülttem az érdeklődésnek, meg is köszöntem, és felhasználtam részeit a módosításnak, leginkább a tracing crate használatát, az Observer modult, illetve a függősségek frissítését. A tracing crate-hez inicializálni kellett a könyvtárat, hogy kiírja az üzeneteket, illetve a `println!` makrókat le kellett cserélni a `debug!` makróra, hogy a kiírást a könyvtáron keresztül végezze.

Ez egyébként nem is az első alkalom volt, hogy egy ismeretlen felhasználó küldött pull request-et, még tavasszal kaptam egyet LMH01-től, ami lehetővé tette, hogy a bemeneti fájlt parancssori argumentumként adjuk át a programnak. Ezt össze is vontam³⁴ (merge) akkor, így a kódban a parancssori argumentumok feldolgozásához használt könyvtár bevonása, és az Args struct létrehozása az ő műve. Ezúton is köszönöm mindenkit a projekthez történő hozzájárulásukat. Azóta persze én is adtam hozzá további parancssori opciókat.

4.2.2 Wave

Ez a modul kezeli a hullámoss hangfájlokat. Van fájlnévből a babycat nevű könyvtár Waveform típusába fájlbeolvasó függvénye például. A Waveform típus képes hullámforma hangfájl eltárolására: eltárol egy minták tömbjét, a mintavételezési frekvenciát, és a csatornák számát, ami jelen esetben mindig 1 lesz, mivel a projektben csak egysatornás (mono) hanggal dolgozunk. A mono-vá történő konverzióra van lehetőség a betöltéskor.

Itt írtam meg a hullámforma hangfájl hangmagasság-változtatásának függvényét is, aminek a matematikáját a 3.1.3 fejezetben részleteztem. Emellett két hangfájlt is össze lehet adni, a másodikat egy adott, nemnegatív egész számmal eltolva. Itt csak a minták egyszerű

³³<https://github.com/4321ba/mp3-to-nbs/pull/3>

³⁴<https://github.com/4321ba/mp3-to-nbs/pull/2>

összeadása történik, az eltolás figyelembe vételével, illetve a túlindexelések helyén 0, azaz csönd alkalmazásával.

4.2.3 Fourier

Ebben a modulban található az FFT meghívása, a spektrogramok kiszámítása, illetve a spektrogramok közötti műveletek.

Először is a Fourier-transzformációt kiszámoló függvény először egy, a spectrum-analyzer könyvtárban definiált Hann ablakfüggvényt alkalmaz a bemeneti, idődimenziójú mintákra, majd meghívja a complex_spectrum modul transzformáló függvényét. A minták f32 szelet (slice) referenciájaként érkeznek, ami azt jelenti, hogy egy ismert hosszúságú, memória-folytonos, nem módosítható helyen vannak a bemeneti adatok, 32-bites lebegőpontos számként. A kimenet egy 32-bites lebegőpontos számokkal dolgozó komplex számokból álló vektor, azaz dinamikusan nyújtható, folytonos memóriaterületet felhasználó tömb adatstruktúra. Ezeket a továbbiakban nem fogom ilyen részletességgel kifejteni.

```
fn transform_fourier_complex(samples: &[f32]) -> Vec<Complex32> {
    let hann_window = hann_window(samples);
    samples_fft_to_complex_spectrum(&hann_window)
        .expect("Something went wrong with calculating fft")
}
```

Itt írtam meg a spektrogram készítő függvényt is, ami szintén lebegőpontos mintákat kap bemenatként, és ugrásonként (hop size) meghívja a Fourier-transzformációs függvényt a megfelelő méretű mintákkal (fft size). Ahhoz, hogy a minták elején levő adatot se veszítsük el, először kibővíti fft_size felének megfelelő csönddel (0-val) a minták elejét és végét. Így az alább látható szegmensekből készül Fourier-transzformáció, amiknek az eredményét tömbként, azaz komplexeket tartalmazó, 2 dimenziós tömbként adja vissza. Ez a komplex spektrogram.

A következő, apró példán minden karakter 1-1 lebegőpontos mintának felel meg, az fft-méret 32, az ugrásméret 8.

```
-----csend-----xxxxxxxxxxxxxxxxxxxxxxxxxxxxx-----csend-----
\-----|-----|-----|-----/|-----|-----|-----|-----|
 \-----|-----|-----|-----/|-----|-----|-----|-----|
 \-----|-----|-----|-----/|-----|-----|-----|-----|
 \-----|-----|-----|-----/|-----|-----|-----|-----/|-----
```

Mivel az fft-méret, azaz az fft-nek átadott minták darabszáma 32, ezért az X-szel jelölt tényleges minták elő 16 mintányi csöndet szűrunk be. Így az első spektrumnak (az FFT

számítás kimenetének) közepe éppen a 0 időpillanatnak fog megfelelni. Ezután az ugrásméret mondja meg, hogy hány mintával toljuk arrébb az egészet, hogy a következő Fourier-transzformációt elvégezzük. Ebben a példában ez 8, tehát a csend közepétől a 24. mintáig fog tartani a következő tartományt. Ugyanígy, az ábra alsó felén levő tartományok jelzik az FFT bemenetének megfelelő mintákat. A kimenet időirányú hossza tehát 5 lesz, azaz 5 FFT eredménye kerül visszaadásra. A frekvenciabeli felbontás függ az FFT méretétől, de mivel a bemenet valós, ezért és a Fourier-transzformáció tulajdonságai miatt a könyvtár csak a felét számolja ki az eredménynek, mert ezzel is a teljes információt meg tudja adni. A függvényenél még lehetőség van limitálni az időbeli kiterjedést egy ugrásszám paraméterrel, amennyiben a minták spektrogramjának csak az elejére van szükség.

A modulon kívülről is elérhető az ezeket csomagoló függvény, amelyik egy hullámforma hangfájl típusú változó komplex spektrogramját képes visszaadni. A projekt során használt paraméterek a 4096-os FFT méret, mert az még nem mossa el az időbeli kiterjedést túlzottan, viszont a mély hangoknál már elegendő frekvenciabeli felbontást nyújt, illetve az 1024-es ugrásméret. Azért az 1024-re esett a választásom, mert az ablakfüggvény alkalmazása miatt a 4096 mintának leginkább a középső része jut érvényre, ezért 4096-os ugrásméretnél értékes információt vesztenénk a két transzformáció határánál, 1024-nél viszont nem érdemes alacsonyabbra menni, mivel akkor a szomszédos transzformációk eredménye már kellően tartalmazza azokat a mintákat, amiből az aktuális transzformáció történt.

Több helyen továbbra sem a komplex spektrogramra van szükség, hanem az amplitúdó spektrogramra, azaz a 2 dimenziós komplex tömbben szereplő komplex számok normái által alkotott 2 dimenziós tömbre. Ehhez a konverzióhoz is írtam egy függvényt.

A modulban találhatóak még két amplitúdó- illetve komplex spektrogram különbségét, illetve összegét számoló algoritmusok, amelyek figyelnek arra, hogy 0-kkal helyettesítésük a spektrogramot, amennyiben az egyik kevésbé széles, mint a másik. Illetve van még amplitúdó- illetve komplex spektrogramok távolságát meghatározó függvény is, aminek egy távolságmérték függvényt kell átadni, majd az algoritmus elemenként meghívja az átadott függvényt, és az eredményt összegzi, egyetlen lebegőpontos számként.

4.2.4 Debug

Ez egy egyszerű modul, a fejlesztés során tette lehetővé számomra a Waveform típus wav fájlba írását, illetve amplitúdó-spektrogramok képfájlba írását.

4.2.5 Note

Ebben a modulban találhatók a felismerendő hangminták fájlnevei. A tömb indexe határozza meg a hangszerhez rendelt számot a programon belül. Itt hoztam létre a `CachedInstruments` típust is. Ez eltárolja egyrészt a beolvasott hangmintákat és a megnyújtott, illetve lerövidített változataikat, hogy ne kelljen minden beolvasni és újramintavételezni. Ezek egy vektorok tömbjében vannak eltárolva, először a hangszer számával, majd a hangmagassággal (0 és 24 között, inkluzív) kell indexelni, hogy megszerezzük a Waveform típust. Ugyanígy a gyorsítótár típusban el tudok tárolni komplex és amplitúdó spektrogramokat is, ugyanúgy kell indexelni őket, mint a hullámformákat, hogy megszerezzük az adott hangszer-hangmagasságához tartozó hangminta spektrogramját.

A `cache_instruments` függvény megkapja a hangfájlok mappáját, és visszaad egy `CachedInstruments` példányt. A szükséges számítások elvégzéséhez meghívja a `wave` és a `fourier` modulok megfelelő függvényeit. A példány tulajdonjogát átadja a meghívó függvénynek, ami így módosíthatná is akár, de ez igazából felesleges, mivel futás közben már nem fog változni a gyorsítótár. Így ha a hívó nem megváltoztatható változóban tárolja el a `cache-t`, akkor ezután akármennyi szálnak átadhatja párhuzamosan, és nyelvileg garantálható, hogy nem lesz probléma az egyszerre történő adatelérésből.

A Note típusban el tudom tárolni egy Note Block felismeréshez szükséges adatait, azaz a hangszer ID-jét és hangmagasságát (előjel nélküli egész számok), illetve a hangerőt (32-bites, lebegőpontos szám). A hangerő általános esetben 0.0 és 1.0 között lehet, de amennyiben egy időpillanatban ugyanaz a hangszer ugyanazon a hangmagasságon többször is szól, akkor azok hangerőinek összege lesz, ami így lehet 1.0 fölött is. Ez kifejezetten gyakori a teszt adathalmazban, a fő dallam kiemelése céljából.

Végül úgy döntöttem, hogy a hullámformákat hangok szelete alapján összeadni képes függvényt is itt helyezem el, és nem a `wave` modulban. Ez a függvény Note-okat kap bemenetként, a hullámformáik közül meghatározza a leghosszabbat, létrehoz egy akkora puffert, majd hozzáadja a hullámformákat, a hangok hangerejével súlyozva. Érdekes és fontos kérdés, hogy a túlvezérlést szimuláljuk-e, vagy sem. Az itt úgy jelenne meg, hogy a hullámforma végső eredményét -1.0 és 1.0 közé szorítjuk. A túlvezérléssel adatot veszítünk, ugyanakkor sok való életből szedett Note Block-os zene túl van vezérelve. Az volt az elképzelésem, hogy ha itt szimulálom a túlvezérlést, akkor közelebb kerülök a túlvezérelt eredeti zene spektrogramjához. Ez nem feltétlenül egy rossz gondolat, de azért, hogy a

spektrogramok lineáris kombinációi megegyezzenek a hullámformák lineáris kombinációjának a spektrogramjával (lineáris tulajdonság) elvetettem.

Ugyanígy található egy spektrogramokat hangok alapján súlyozottan összeadó függvény is. A használatot kényelmesebbé teendő ennek van egy lebegőpontos szelet paramétere is, hangerő-felülírásra, hogy ne kelljen a hívónak összekombinálni a hangerőket a Note példányokkal. Ezt a kombinációt a függvény elvégzi jelenleg, hogy könnyebb legyen a hangokon iterálni, de tulajdonképpen elég volna a hang hangereje helyett a felülírást alkalmazni. A jelenlegi megoldással viszont meg lehetne hívni a függvényt felülírás nélkül is, és akkor a Note-ok hangerejét használja.

4.2.6 Nbs

Ebben a modulban történik a fájlba írás előkészítése, illetve a fájlba írás. Van egy konstans leképezés, ami a hangszerek fájlneveit képezi a fájlformátumot kezelő könyvtár által definiált felsorolt típus értékeire. Ez egy picit bonyolult megoldás, mivel így egy-egy hangszerhez 2-féle számérték tartozik, egy a saját programban használt ID, egy pedig a könyvtárban definiált. Ugyanakkor véleményem szerint ez a megfelelő megoldás, mivel így könnyen ki-be tudom kapcsolni egy-egy hangszer felismerését, és így is garantálható, hogy bármilyen hangszer-kombináció esetén 0 és hangszerszám közé esnek az ID-k, azaz lehet velük memória-folytonos tömböt, illetve lyukak nélküli vektort indexelni. Ha pedig az exportáláskor szükség van a könyvtár által definiált enum értékre, először a saját ID alapján meg lehet szerezni a fájlnevét, majd a fájlnév alapján a felsorolt értéket. Azért nem egyből a saját ID-t képzem le az enum értékekre, mert az nem lenne stabil, amennyiben megváltoztatom a felismerendő hangszereket, a többi ID-je is változhat, így több helyen lenne szükséges módosítani. Továbbá, ha a jövőben futási idejű konfigurációt is lehetővé szeretnénk tenni, például parancssori paraméterrel meghatározva a felismerendő hangszereket, akkor is könnyebbség ez a megoldás.

A felismert Note példányok közül van sok, ami közel 0 hangerejű, ez annak felel meg, hogy a hang ott valójában nem szól. Ezentúl lehetnek olyan hangok is, amik 1-nél nagyobb hangerejűek. Ezek megtisztítására írtam egy függvényt, ami egy meghatározott érték alatt (például 0.15) eldobja a hangot, illetve az 1-nél nagyobb hangerejű hangpéldányokat felbontja 1 hangerejű, illetve a törtrész hangerejű hangpéldányokra. Ebben a modulban szükséges figyelni, hogy a megfelelő Note struktúrát használjuk, mivel az nbs könyvtárban is található egy note modulon belüli Note osztály, amit az exportáláshoz használni kell. Szerencsére a

fordító szól, ha valamely típusok azonos nevűek, de igazából két külön típus, és emiatt összekevertem őket, és így inkompatibilis, amit írtam. A megoldás egyszerű, ki kell írni a teljes útvonalat a típushoz, és akkor egyértelmű lesz, vagy egyedibb típusnevet is lehetne használni.

Itt írtam meg az `export_notes` függvényt, a könyvtár példájának³⁵ felhasználásával. Paraméterként kap egy hangok két dimenziós tömbjét, amiben az exportálandó Note Block-ok vannak a felismerés eredményeként, idővel indexelve meg lehet szerezni az abban a pillanatban szóló hangokból álló vektort. Szokás, és szébb, ha az NBS fájlban egy-egy sorban (layeren) csak ugyanolyan hangszerű hangok vannak. Ennek érdekében meg kell határozni hangszerenként az egy időpillanatban egyszerre szóló hangok maximális számát. Ezt csinálja a következő kód részlet.

```
let layer_counts_by_instrid: Vec<usize> =
    (0..note::INSTRUMENT_COUNT).map(|instr_id|
        notes.iter().map(|tick|
            tick.iter().filter(|note| note.instrument_id == instr_id).count()
        ).max().unwrap()
    ).collect();
let max_layer_count = layer_counts_by_instrid.iter().sum();
```

Kiindul az összes érvényes hangszer ID-ből, és ezt képezi le az adott hangszerhez szükséges layerek számára. A leképzés úgy történik, hogy a hangok összes időpillanatán keresztülmegy, és megszámolja az adott időpillanatban a megfelelő hangszerű hangokat. Így minden hangszer hangjai el fognak férfi a saját soraikban, az összes szükséges layer száma pedig ezek összege.

Ezután a Note Block-okat úgy kell lepakolni tick-enként, hogy megfelelő mennyiségű szünetet hagyjon ki, ha hangszerváltás történik. Nem szükséges, hogy a hangok 2 dimenziós tömbjének idődimenziója minden tick-et tartalmazzon, erre van a `timesteps` paramétere a függvénynek. Ez megmondja, hogy az i-edik vektor a kétdimenziós tömbben hányadik tickben játszódik.

A hangerőt 100-zal szorozzuk, mert az NBS fájlformátumban 0 és 100 közötti a hangerő. Továbbá túlnyomórészt csak 10%-onként változtatják a hangerőt a zeneszerzők, mert az is elegendő kontrollt ad, illetve a felület azt teszi a legkönnyebben lehetővé. Emiatt én is 10-enként kvantálom a kimeneti hangerőt.

³⁵<https://crates.io/crates/nbs-rs>

4.2.7 Tempo

A tempo modulban található a ritmusfelismerés, a hangkezdés-felismerés, és az ezekhez tartozó függvények. Az aubio könyvtár segítségével lehet tps-t (tick per second, a tempó), illetve onseteket (hangkezdéseket) felismerni. Ezután ezeket különböző-féle heurisztikákkal feldolgozó függvényeket is írtam. A hangfelismerés pontosságának méréséhez viszont fel fogom használni az eredeti fájlból a tps információt, mivel ez egyetlen szám, és amennyiben mellétippli a rendszer, az egész felismerés használhatatlan lesz. Illetve, ha kétszerest, vagy a felét tippeli (ami gyakran előfordul, és nincs igazán jó módszer a kiküszöbölésére), akkor az eredeti hangfájllal lesz az összehasonlítás kicsit nehezebb.

Továbbá az nbswave kiszámítható pontossága miatt lehet tudni, hogy az onsetek is tökéletesen egyenlő távolságra fognak történni, ami kiszámítható a tempóból, illetve az első onset rögtön a 0 indexű mintánál fog történni. Mivel a célként kitűzött mérést az nbswave-vel generált hangfájlokon fogom elvégezni, így ezeket az információkat fel tudom használni a mérések pontosítása érdekében. Ezek miatt a modul tartalmának nincs nagyon nagy jelentősége. Meg fogom viszont a hangfelismerés pontosságától függetlenül a tps-felismerést is mérföldet az 5.3 fejezetben, hogy hány esetben találta el pontosan a tempót, illetve milyen módon hibázik gyakran egy nagyobb adathalmazon.

4.2.8 Optimize

Ez a modul tartalmazza a módszer lelkét. A `full_optimize_timestamp` függvény a belépési pont a modulba, amely bemenetként kapja a hangserek gyorsítótárát, az optimalizálandó zene hullámformáját, az eddig felismert zene hullámformáját (a felismert hangok hullámformáinak megfelelően eltolt összegét), egy hangkezdés-időpontot hangmintákban számolva, illetve a tempót. Ezek alapján visszaadja az adott időpillanatban felismert hangok vektorát.

Az első teendő a két hullámformából a szükséges apró rész, a hangkezdés időpontjától rövid ideig tartó részlet kivágása, és abból egy új hullámforma készítése. Szükséges lekezelni azt az esetet is, hogyha a hullámforma nem elég hosszú, esetleg a felismert hangok hullámformája a kezdeti pillanatig sem feltétlen tart. Ezekből a részletekből aztán spektrogramokat készítünk. Itt nagyon fontos közölről megvizsgálni a sorrendet. Egy korábbi verzióban a korábban felismert hangoknak nem a hullámformáját kapta a függvény, hanem a spektrogramját, és az közel sem eredményezett ilyen pontos felismerést. A fázisról ennek a fejezetnek a költségfüggvény részénél fogok részletesen írni, de kiemelkedően fontos, hogy a

spektrogram készítése mintára pontosan az onset-től kezdődjön, mivel a minták spektrogramja is pontosan innen kezdődik. Ez ismételten felhasználja azt, hogy az nbswave precízen exportál.

Történik ezután egy előfelismerés (`test_distances_for_instruments` függvény), ami ad egy potenciálisan szóló hangok vektorát, így kiszűrve nagyon sok hangszer-hangmagasság párt, ami biztosan nem szól az adott pillanatban, ezzel sokat csökkentve az állapottéren. Majd ennek a vektornak a pontosítása érdekében alkalmazok egy könyvtári minimalizáló algoritmust (`optimize` függvény).

Az előfelismerés azt csinálja, hogy aszimmetrikus távolságot számol a felismerendő zene, és az adott hangszer-hangmagassághoz tartozó minta amplitúdó spektrogramja között a következő módon: a hangmintát először is lehalkítja, hogy csendesebben szóló hangokat is felismerjen, például 0.2-szeresére. Ezután kivonja a zene spektrogramjából a hangminta spektrogramját, majd kizárolag a negatív számok négyzeteit összeadja, ez lesz a hiba. Hasonló módon kivonja a csendből is a halkított hangmintát, ez lesz a referenciahiba. A kettő hányadosa ad egy értéket, hogy az adott hangminta mennyire biztosan szerepel a zene ezen pillanatában. Ha ez egy bizonyos érték alatt van (például 0.035 alatt), akkor azt mondhatjuk, hogy az adott hangszer-hangmagasság pár lehet, hogy szerepel ebben a pillanatban, és ezt érdemes közelebbről megvizsgálni. A kódban a tényleges számítások során nem pontosan ilyen függvényhívások történnek, de a lényegi algoritmus ez.

A könyvtári optimalizációs lépéshez először definiálni kell egy típust, ami eltárolja az optimalizációhoz szükséges paramétereket. Ez jelen esetben a következő.

```
struct OptiProblem<'a> {
    cache: &'a note::CachedInstruments,
    multiplier: f32,
    song_part: &'a [Vec<Complex32>],
    previous_part: &'a [Vec<Complex32>],
    found_notes: &'a [note::Note],
    hops_to_compare: usize,
}
```

Megkapja a gyorsítótárazott hangszereket, egy hangerő-szorzót, a felismerendő zene spektrogramjának megfelelő részletét, az eddig felismert zene spektrogramjának megfelelő részletét, az optimalizálandó hangokat (itt csak a hangszer és a hangmagasság számít), illetve az összehasonlítandó ugrások számát, azaz milyen széles legyen az összehasonlítandó spektrogram. A legutóbbi azért fontos, hogy a következő tick előtt befejeződjön az összehasonlított zenerészlet, így az ne rontsa a felismerést. Az `'a` sablonparaméter az

élettartamot jelöli³⁶, ugyanis a fordító garantálja, hogy a struktúrában szereplő referenciák a struktúra példányának teljes élettartama alatt érvényesek fognak maradni, ehhez viszont szükséges őket explicit módon felparaméterezni. Amennyiben úgy próbálnám használni a struktúrát, hogy a konstruáláskor átadott referenciák élettartama nem volna elég hosszú, a kód nem fordulna.

A könyvtári optimalizáció meghívásáért felelős függvény pedig létrehoz egy példányt ebből a típusból, illetve az optimalizációs módszerből is, felkonfigurálja a futtatásért felelős környezetet például az iterációk számával, meghívja az optimalizációt, majd létrehoz egy új hangokból álló vektort a felismert hangerőkkel. Hasznos hibakeresésnél, ha ez a függvény képként elmenti az adott időpillanatbeli zenerészletet, illetve a zenerészlet és a felismert hangok különbségét, ami alapján a költség is számolódik. Így futás közben a kép nyomon követésével tudom vizualizálni az éppen tesztelt módszer pontosságát: lehet látni, hogy el van-e csúszva a felismerés, van-e olyan hang, amit nem sikerült felismerni, az előzőleg felismert hangok kellően kioltották-e az eredeti zene megfelelő részét, vagy belátszik-e a következő ütem, és attól lett nagyobb például a hiba.

Az optimalizációhoz persze szükség van egy költségfüggvényre is. Ehhez az argmin könyvtár definiál egy költségfüggvény jellemvonást (trait), ami az interfész Rust-beli megfelelője. Ezt tetszőleges struktúrára lehet implementálni, két típust, az optimalizálálandó paramétert és a hiba típusát, és egy függvényt, a hibafüggvényt kell hozzá implementálni.

```
impl CostFunction for OptiProblem<'_> {
    type Param = Vec<f32>; // it should be found_notes.len() long
    type Output = f32;
    fn cost(&self, param: &Self::Param) -> Result<Self::Output, Error> {
        assert_eq!(param.len(), self.found_notes.len(),
                   "Volume guess vec should be as long as the notes vec to guess");
        let added_spectrogram = note::add_note_spectrograms(
            self.found_notes, param, self.cache, self.multiplier);
        let with_previous = fourier::add_spectrograms(
            &added_spectrogram, self.previous_part);
        let found_part = &with_previous[
            0..std::cmp::min(self.hops_to_compare, with_previous.len())];
        let diff = fourier::calculate_distance_complex(
            self.song_part, found_part, &|sp, fp| (sp-fp).norm_sq());
        Ok(diff)
    }
}
```

A költség számítása pedig úgy történik, hogy összeadom a hangokat a tippelt hangerőn (param függvényparaméter), hozzáadom a korábban felismert hangok lecsengését,

³⁶<https://doc.rust-lang.org/rust-by-example/scope/lifetime/explicit.html>

kivonom belőle a felismerendő részletet, és az így kapott komplex számok normanégyzetét veszem. Ahhoz természetesen, hogy ez ilyen egyszerűen működjön, sok mindenre szükség van, például arra, hogy a komplex számok összeadásakor a fázisok éppen megfelelően oltsák ki, vagy erősítsék egymást. Ezt az garantálja, hogy a spektrogramok kezdetei pontosan az összes hullámforma közös kezdőpillanatában vannak, azaz ugyanaz a pillanat felel meg az onset-nek (az újonnan szóló hangok megszólalásának kezdőpillanatának), mint ahol a korábban felismert hullámforma spektrogramszámításhoz felhasznált része kezdődik, és ez pontosan ugyanaz a pillanat, ahol a felismerendő zenében is az onset történt. Továbbá, mint azt korábban említettem többször, az nbswave kiszámíthatóan pontosan adja össze a hullámformákat, így lehet arra számítani, hogy a felismerendő hangfájlban is pontosan ugyanolyan kioltások és erősítések fognak történni a komplex számok között, mint a korábban felismert hangok lecsengése, és az újonnan megszólaló hangok összegénél. Ellenkező esetben is lehetséges az amplitúdó spektrogramokat összehasonlítani, amihez nincs szükség a spektrogramok fázisaira, viszont nem lesz annyira pontos. Meg lehet továbbá becsülni az onset-eket akkor is, ha nem számíthatóak ennyire pontosan, azonban ezen szakdolgozat keretein belül nem foglalkoztam az ilyen módon becsült eltolás által okozott kioltás pontosságával.

A másik nagy előnye a fentebb taglalt, komplex számokat összeadó költségfüggvénynek, illetve annak, hogy a túlvezérlést nem próbáljuk meg a hullámforma szintjén szimulálni, az az, hogy a költségfüggvény deriváltját is meg tudjuk határozni pontosan. A derivált nélkül is fel tudunk használni optimalizációs eljárásokat, a Nelder-Mead módszert használtam, illetve a Particle Swarm Optimization-t próbáltam azelőtt, mielőtt a deriváltat meg tudtam volna határozni, viszont ezek nem olyan pontosak, nem adnak annyira jó eredményt, mint a deriváltat felhasználni képes algoritmusok. Természetesen a hátrány az az, hogy meg kell határozni hozzá a deriváltat, pontosabban a gradienst.

$$\begin{aligned}
& \mathbf{p} \in \mathbb{R}^n, \quad \forall i \in \{1, 2, \dots, n\}: S_i, P, R \in \mathbb{C}^{f \times h} \\
& cost(\mathbf{p}) = \sum_{elements} \left\| \left(\sum_i p_i \cdot S_i \right) + P - R \right\|^2 \\
& \nabla cost(\mathbf{p}) = \left[\frac{\partial cost}{\partial p_1}(\mathbf{p}), \frac{\partial cost}{\partial p_2}(\mathbf{p}), \dots, \frac{\partial cost}{\partial p_n}(\mathbf{p}) \right] \\
& \frac{\partial cost}{\partial p_i}(\mathbf{p}) = \sum_{elements} 2 \cdot S_i * \left(\left(\sum_i p_i \cdot S_i \right) + P - R \right) \\
& \text{where } A * B = \Re A \odot \Re B + \Im A \odot \Im B
\end{aligned}$$

Ábra 8: Formula a költségfüggvényhez és a gradienséhez

A fentebbi formulában (ábra 8) \mathbf{p} -vel jelölöm a paramétervektort, ami a megtippelt hangerőket jelenti, S_i -vel a paramétervektor i -edik komponenséhez tartozó hang spektrogramját, P -vel a megelőző lecsengés spektrogramját (previous), és R -rel a felismert zene spektrogramját (result). A magasságuk az FFT mérettől függ, a szélességük pedig az összehasonlítandó ugrások száma. A költségfüggvény, ahogy azt fentebb ismertettem, a spektrogramok hangerőkkel súlyozott összege, hozzáadva a megelőző lecsengést, és kivonva belőle a felismert zeneit. Az így kapott komplex mátrixnak vesszük a normanégyzetét elemenként, majd összegezzük az így kapott mátrix elemeit.

A gradienshez meg kell határozni a paraméter összes komponense szerinti parciális deriváltat. Az i -edik parciális deriválthoz a költségfüggvényt, ami lényegében sok négyzetes tag összege, p_i szerint kell deriválni. Mivel a deriválás lineáris, elég a mátrix egy elemét deriválni, majd a deriváltakat összeadni. A normanégyzet deriváltjának meghatározásához tekintsük a komplex szám valós és imaginárius részét külön, jelöljük őket a -val, és b -vel. Ezek függnek p_i -től: $\|a(p_i) + b(p_i)i\|^2 = a(p_i)^2 + b(p_i)^2$. A deriválás után a következő eredményt kapjuk: $2a(p_i)a'(p_i) + 2b(p_i)b'(p_i)$. A valós rész kizárolag a mátrixok megfelelő elemeinek valós részeitől, míg az imaginárius rész az imaginárius részektől függ, mert a komplex mátrix, aminek az elemeinek a normanégyzetét vettük, ismert mátrixok lineáris kombinációja (és a szorzás csak valós számokkal történik), így a valós része a mátrixnak a mátrixok valós részeinek lineáris kombinációja, és ugyanígy a komplex részzel is. A paramétervektor i -edik komponense pedig az i -edik hanghoz tartozó spektrogram valós, illetve komplex részének szorzótényezője, így a valós, illetve komplex rész deriváltja p_i szerint maga a spektrogram adott elemének valós, illetve imaginárius része.

Ebből jön a 8. ábrán látható képlet: el kell készíteni a különbség-spektrogramot (az S_i , P és R lineáris kombinációjából), és azt kell az i -edik spektrogrammal a következőképpen

összeszorozni (* operátorként jelöltem az ábrán): elemenként kell venni a valós részeket, azokat összeszorozni, majd hozzáadni a szintén elemenként vett imaginárius részek szorzatát. A \odot operátort nem én találtam ki, az a mátrixok elemenkénti szorzata³⁷. A valós, illetve imaginárius részét a mátrixnak elemenként értettem. Egy komplex szám imaginárius része valós, tehát $\text{Re}(a+bi)=b$, és nem bi .

Az argmin könyvtár definiál egy gradiens jellemvonást (trait-et) is, aminek az implementálása szükséges ahhoz, hogy meghívassunk derivált alapú minimalizáló algoritmusokat is. Ehhez is tartozik két típus, a paraméter ugyanúgy, illetve a gradiens, ami esetben szintén egy lebegőpontos számokból álló vektor lesz, aminek a hossza szintén az optimalizálandó paraméterek száma. Implementálandó továbbá a gradienst meghatározó függvény, ami paraméterként kapja a paramétervektort, és vissza kell adnia a kiszámolt gradienst a keresési tér adott pontjában. A visszatérési érték továbbá, ugyanúgy, mint a költségfüggvénynél is, igazából egy Result típus, ami egy enum, és a lehetséges értékei az Ok, és az Error. Mindkettőbe lehet csomagolni egy-egy értéket, és ennek használatával lehet jelezni a hívó számára, hogy a függvény sikeresen lefutott, vagy hibára jutott.

```
impl Gradient for OptiProblem<'_> {
    type Param = Vec<f32>; // it should be found_notes.len() long
    type Gradient = Vec<f32>;
    fn gradient(&self, param: &Self::Param) -> Result<Self::Gradient, Error> {
        assert_eq!(param.len(), self.found_notes.len(),
                   "Volume guess vec should be as long as the notes vec to guess");
        let added_spectrogram = note::add_note_spectrograms(
            self.found_notes, param, self.cache, self.multiplier);
        let with_previous = fourier::add_spectrograms(
            &added_spectrogram, self.previous_part);
        let subtracted = fourier::sub_spectrograms(
            &with_previous, self.song_part);
        let found_part = &subtracted[
            0..std::cmp::min(self.hops_to_compare, subtracted.len())];
        let grad = self.found_notes.iter().map(|note| {
            let note_spectrogram =
                &self.cache.complex_spectrograms[note.instrument_id][note.pitch];
            let cut_note_spectrogram = &note_spectrogram[
                0..std::cmp::min(self.hops_to_compare, note_spectrogram.len())];
            let diff = fourier::calculate_distance_complex(found_part,
                &cut_note_spectrogram, &|fp, no| fp.re * no.re + fp.im * no.im);
            2.0 * diff
        }).collect();
        Ok(grad)
    }
}
```

³⁷https://en.wikipedia.org/wiki/Hadamard_product_%28matrices%29

A gradiens meghatározását kellő részletességgel taglaltam fentebb, így annak kiszámolásának a módját nem részletezem a kód kapcsán. Fontos megemlíteni, hogy a spektrogramokat megfelelő méretűre kell vágni, hogy ne hasonlítson össze irreleváns részeket, amiben benne vannak már a következő hangok, vagy korábbi levágás miatt csönd van ott, ugyanakkor figyelni kell arra is, hogy amennyiben nem volna a spektrogram elegendően hosszú, akkor se szeleteljünk túl a határán.

Itt egyébként érdekes megfigyelni, hogy a visszatérési érték konkrét típusát, hogy vektorba szeretném összegyűjteni, nem határoztam meg sehol, azt a fordító onnan találta ki, hogy tudja, minek kell lennie a függvény visszatérési értékének. Ha nem használnám fel visszatérési értékként, illetve más módon, amiből a típus kitalálható volna, sem, akkor szükséges lenne valamilyen típus-annotáció: vagy let `grad: Vec<f32>` az elején, vagy az összegyűjtéskor `.collect::<Vec<f32>>();`. Itt jegyzem még meg, hogy Rust-ban nem szükséges kiírni a return-t a függvény végére, viszont ilyenkor pontosvesszőt sem szabad tenni. Ugyanígy lehet if-else konstrukcióban is értéket visszaadni, kiváltva a ?: hármas operátort, csak a típusaiknak meg kell egyeznie.

4.2.9 Main

A main modulban kap helyet a parancssori argumentumokat tartalmazó struktúra, illetve azok feldolgozása a main függvény, azaz a programba történő belépési pont elején. Az egyik húzási kérelem a struktúrát tette egy cli modulba, míg a másik a main függvény tartalmát is különválasztotta egy lib modulba, amit a belépési pont egyből meghívott. Én végül ezeket a main modulban egyesítettem, mivel véleményem szerint létezik olyan modul, illetve olyan függvény, ami túl pici, és szerintem növeli az átláthatóságot, ha ezeket egyesítjük. A main modul így is mindenkel együtt körülbelül 80 soros, ezt nincs szükség 3 részre bontani. A lib modulnak olyan szempontból lehet értelme, hogyha könyvtárként szeretnénk a programot rendelkezésre bocsátani, akkor a fő algoritmust is könnyen meg lehet hívni.

A main függvény az argumentumok feldolgozása után betölti a bemeneti hangfájlt, meghatározza a tempót és az onset-eket, majd gyorsítótárazza a hangmintákat, és spektrogramjaikat. Ezután egy ciklus végigmegy az összes onset-en (azaz potenciális hangkezdet pillanaton), meghívja az optimalizációt, majd a cikluson kívüli gyűjtőkbe beleteszi a megtalált hangokat, illetve a hullámformáikat hozzáadja a korábban felismert hullámformákhoz. A következő iterációban ezt a hullámformát adja át az optimalizációnak,

mint a korábban megtalált hangok lecsengése. Ezzel a módszerrel a korábbi lecsengést nem csak a legutóbbi ütem hangjaiból számolja, hanem az összes korábbiból is, amennyiben még ténylegesen tart ott a hang. Az iterációk közben ír helyzetjelentést a konzolra, illetve a debug! makróval részletesebb leírást is, hogy lehessen tudni, hol jár a program. Ezután a megtalált hangokat NBS-ként exportálja a megadott kimeneti fájlba.

5 Önálló munka értékelése, eredmények

5.1 Értékelés módszere

Az értékeléshez célszerű sokféle szempontot figyelembe venni. Ebben a fejezetben több szempontból megvizsgálom a program által adott eredményt, illetve a programot. Egyrészről megvizsgálom a célul kitűzött exportálási forma (nbswave) által adott hullámos hangfájl felismerésének pontosságát mind hangfelismerés szempontjából, mind a tempó felismerésének szempontjából. Ezután összevetem az eredményt azzal, amit egy eddig is elérhető, automata módszer produkálni volt képes. Végül kitérek a program jelenlegi verziójának fontos negatívumára, a felismerés robusztusságára

5.2 Hangfelismerés pontossága

5.2.1 Tesztkörnyezet

A pontosság méréséhez létrehoztam egy kis tesztkörnyezetet, ami a projekt mellett ugyanúgy elérhető a GitHub-on³⁸. A tesztelés alapjául a 2.2.5 fejezetben részletesebben kifejtett WynnCraft Noteblock OST adathalmaz szolgált, kiegészítve a régebb óta példaként felhasznált Eight-Legged Foe-val[23]. Fontos megemlíteni, hogy nem a WynnCraft szerver eredeti hangmintáit fogom használni a teszteléshez, amikhez a zenéket írták, hanem a Minecraft eredeti hangmintáit. Ez az adathalmaz összesen 224 darab NBS fájl. Ezeket átkonvertáljuk hullámforma fájlokká, méghozzá olyan hangmintákkal, amiket 20dB-lel lejebb halkítottam, így elkerülve a túlvezérlést. Az nbswave képes a túlvezérlést kompenzálni, és ennek érdekében lejebb halkítani az exportált zenét, ennek a mértékét viszont kifejezetten nehéz lenne a hullámforma fájlból megbecsülni, így amellett döntöttem, hogy a túlvezérlést elkerülendő az eredeti hangmintákat halkítom. Az nbswave egy Python könyvtár, így készítettem egy rövid szkriptet, amivel a parancssorból meghívható lesz.

```
#!/bin/env python3
# required: pip3 install nbswave==0.3.0
from nbswave import render_audio
from sys import argv
print(f"Converting {argv[1]} to {argv[2]} with sounds {argv[3]}")
render_audio(argv[1], argv[2], default_sound_path=argv[3])
```

³⁸https://github.com/4321ba/mp3-to-nbs/tree/main/WynnCraft_Noteblock_OST

A 0.3.0 verzió fontos, nekem a 0.4.0, a jelenlegi legfrissebb verzió nem műköött. Jelentettem a potenciális hibát a fejlesztőnek GitHub-on³⁹, aki ezidáig még nem válaszolt.

Ezután ezt kell meghívjuk az összes nbs-beli fájlra, és exportálni a wave mappába. Mivel ez már eltarthat egy ideig, utána nézem, hogy hogyan lehet egyszerűen bash-ben párhuzamosítani anélkül, hogy az összes folyamatot egyszerre indítanánk el⁴⁰.

```
#!/bin/bash
N=16
(
for f in nbs/*
do
((i=i%N)); ((i++==0)) && wait
./nbswave_runner.py "$f" "wave/"`basename $f .nbs`.ogg" "../SoundsQuiet" &
done
)
```

Ez a szkript az nbs-beli fájlokon iterál, és a háttérben elindítja a konverziót a fentebbi Python szkripttel. Viszont minden 16. iterációban meghívja a wait parancsot, ami megvárja, hogy az összes háttérfolyamat befejeződjön. Ez nem használja ki tökéletesen az elérhető erőforrásokat, mivel lehet, hogy az egyik zene hosszabb, és a következő adagot nem fogja a szkript elindítani, amíg az befejeződik, viszont a célomnak megfelel.

Ezután, ezzel analóg módon történik a szakdolgozat programjának meghívása is, annyi különbséggel, hogy átadom neki a tempót az eredeti NBS fájlból. Ezt azért teszem, mert a 4.2.7 fejezetben taglaltakhoz kapcsolódóan ez az egyetlen szám nagymértékben tudja befolyásolni a felismerést, azaz ha eltalálja a program, akkor nagyon jó felismerést képes elérni, viszont, ha nem találja el, akkor felesleges bármivel is próbálkozni. Emiatt arra jutottam, hogy a hangfelismerést és a tempófelismerést külön tesztelem, és a hangfelismerésnél feltesszük, hogy a tempófelismerés sikeres volt.

Ezután írtam egy egyszerű Python szkriptet, ami összehasonlít két NBS fájlt a tartalmuk alapján. Mivel a tempó elég egyértelmű, szerencsére időbeli elcsúszásra nem kell felkészülni.

```
def fill_list_from_song(volume_list, song):
    for tick, chord in song:
        for note in chord:
            if note.key < 33 or note.key > 57:
                continue
            volume_list[tick][note.instrument][note.key - 33] +=
                note.velocity / 100 * song.layers[note.layer].volume / 100
```

³⁹<https://github.com/OpenNBS/nbswave/issues/7>

⁴⁰<https://unix.stackexchange.com/questions/103920/parallelize-a-bash-for-loop>

A szkript kigyűjti az előbbi függvényekkel minden NBS fájlból az adott időpillanatban, adott hangszeren, adott hangmagasságon szóló Note Block-ok hangerőinek összegét. Az összegyűjtés után többféle metrikát készít a szkript: veszi a két zene megfelelő időpillanat-hangszer-hangmagasság kombinációinál található hangerőik különbségeinek négyzetösszegét (hangerőhiba), majd referenciaként veszi az eredeti hangfájl hangerőinek négyzetösszegét is (csend hibája). Ebből tudok egy százalékos hibát számolni: a két szám hányadosát kell 1-ből kivonnom. Így a 100% felel meg a tökéletes felismerésnek, míg a 0% a csenddel egyenértékű hibának. A felismerés könnyen tud 0% alá is menni, ha több helyre történt rossz hang betétele, mint jó. Ugyanakkor mégis ezt tartom a legjobb százalékértéknek, mivel figyelembe veszi a hangerőt is, illetve egy halk hang félreismerését nem bünteti annyira, mint egy hangosét, ezenkívül az elméleti legrosszabb felismeréshez nincs értelme viszonyítani a 0%-ot, mivel olyan hatalmas az állapottér, hogy a csend is túlnyomórészt több, mint 90%-ot érne el.

Egyéb, beszédes statisztikaként felsorolom az eredeti hangok számát (pozitív), a jól felismert hangok számát (valós pozitív), a nem felismert hangok számát (álnegatív), a félreismert csendek számát (álpozitív), illetve a jól felismert csendek számát (valós negatív) is⁴¹. Ezekből is lehet százalékos értékeket képezni, az eltalált csendek óriási száma miatt azok a beszédedesebb mutatók, amik azt nem tartalmazzák. A precizitás megmondja, hogy a tippelt hangok mekkora része volt tényleges hang, míg a szennitivitás azt árulja el, hogy a hangok hány százaléka lett eltalálva.

5.2.2 Változtatható paraméterek ismertetése

A hangfelismerés során nagyon sok paraméter változtatása tud javítani, illetve rontani a felismerésen. A legfontosabb kérdés az optimalizáló algoritmus. Deriváltmentes algoritmusok közül előző félévben megvizsgáltam a Nelder-Mead módszert, illetve a Particle Swarm Optimization-t, amik közül a Nelder Mead került ki egyértelműen gyorsabbnak, és pontosabbnak⁴². Ebben a félévben, mivel elérhető a derivált, a Steepest descent-tel, illetve a Nonlinear conjugate gradient módszerrel is megismerkedtem. Ezentúl változtatható még a felismerés során az iterációk száma, a kezdőpozíció, a derivált alapú algoritmusoknál a vonalmenti keresés algoritmus, illetve a felismerendő hangszerek száma is.

⁴¹https://hu.wikipedia.org/wiki/Pontoss%C3%A1g_%C3%A9s_precizit%C3%A1s

⁴²<https://github.com/4321ba/mp3-to-nbs/blob/main/documentation/paper/documentation.md#optimaliz%C3%A1l%C3%A1si-algoritmusok-%C3%B6sszehasonl%C3%ADt%C3%A1sa>

A vonalkeresések közül kettő elérhető, a More-Thuente, illetve a Hager-Zhang. A második implementációja hibát dobott azzal a szöveggel, hogy elérhetetlen pontot ért el az algoritmus, megkérve, hogy jelentsem GitHub-on⁴³, így maradt az első opció.

5.2.3 Alapdob

Az adathalmazban 5-féle hangszer van: hárfa, basszusgitár, kíkk, pergődob és alapdob. Ezekhez minden tartozik 25-féle hangmagasság, és az alapdobot a legnehezebb felismerni, mivel elég egyenletes a spektruma, és nehéz a hangmagasságai között különbséget tenni. Az alábbiakban az Eight-Legged Foe-n futtattam Steepest descent algoritmust, 80-as iterációsámmal, először 0.5-ös kezdeti hangerővel alapdob nélkül, majd 0.0 illetve 0.5-ös kezdő hangerővel, alapdóbbal. Az alapdobot felismert fájloknál kétszer futtam az összehasonlítást, egyszer az alapdobot figyelembe véve, egyszer nem.

Táblázat 1: Alapdob felismerése

Kezdeti hangerő	0.5	0.0	0.5	0.0	0.5
Alapdob felismerése	Nem	Igen	Igen	Igen	Igen
Alapdob összehasonlítása	Nem	Nem	Nem	Igen	Igen
Csend hibája	792.86	792.86	792.86	860.7	860.7
Hangerőhiba	0.76	14.98	7.58	64.46	56.72
Eredeti hangok száma	2141	2141	2141	2565	2565
Jól felismert hangok száma	2137	2139	2139	2499	2499
Nem felismert hangok száma	4	2	2	66	66
Csend hangként felismerése	2	60	78	238	287
Jól felismert csend	106757	106699	106681	133322	133273

Lehet látni, hogy a 424 alapdóból 358-at jól felismert (2.-3. és 4.-5. oszlopok különbsége), viszont a többi hangszer felismerésén is sokat rontott a jelenléte. Továbbá a felismerése nélkül a hangerőhiba és a felismerés szinte tökéletes, míg ha hozzáadom az alapdob felismerését, elég sok csend helyére tippel hangot, illetve a hangerőhiba is jelentősen romlik. Ezeken valószínűleg lehetne javítani többek között az iterációsáam növelésével, illetve az előfelismerés pontosításával, de a további tesztelést négy hangszerrel végzem, azaz

⁴³<https://github.com/argmin-rs/argmin/issues/541>

a szintetizált hullámforma fájlból, a felismerésből és az összehasonlításból is kihagyom az alapdobot. Továbbá lehet látni azt is, hogy a 0.0 és a 0.5 kezdeti hangerő között nincs nagyon sok különbség, de a 0.0 kevesebb nem létező hangot tippelt be, viszont a 0.5 hangerőhibája jobb. A 5.2.6 fejezetben még közelről megvizsgálom a két lehetőséget.

5.2.4 A Nelder-Mead módszer és a Steepest Descent összehasonlítása

A Nelder-Mead módszerrel futtattam 3 egyszerű zene felismerését, majd ezek után a Steepest Descent-tel is felismertettem ugyanazokat. A Nelder-Mead felismerése jóval gyorsabb volt, viszont a Steepest Descent sokkal pontosabb. A 2-es táblázat ezt foglalja össze. A három zene a [001] Luxury of the Cease-Fire (Ragni), [007] Saddle Up (Ternaves), illetve a [026] Yearning for the Days of Glory (Ancient Nemract) voltak. Az eredmények miatt úgy döntöttem, a továbbiakban a deriváltat felhasználni képes algoritmusokkal folytatom a vizsgálatot.

Táblázat 2: A Nelder-Mead módszer, és a Steepest Descent összehasonlítása

Zene	001	001	007	007	026	026
Módszer	NM	SD	NM	SD	NM	SD
Csend hibája	722.56	722.56	3020	3020	1357.6	1357.6
Hangerőhiba	100.86	0.01	508.64	0.3	317.97	3.67
Eredeti hangok száma	1039	1039	1344	1344	1372	1372
Jól felismert hangok száma	1022	1039	1326	1344	1347	1369
Nem felismert hangok száma	17	0	18	0	25	3
Csend hangként felismerése	438	0	1423	0	1376	2
Jól felismert csend	113823	114261	99533	100956	106152	107526

A Nelder-Mead is felismerhető eredményt adott, viszont jól látható, hogy néhol több rossz hangot tippelt be, mint jót. A hangerő pontossága is a csendhez viszonyítva jóval 90% alatt van. Ráadásul az optimalizáló sokszor konvergált is, így az iterációszám növelésével nem lehet könnyen javítani rajta. Ezzel szemben a Steepest Descent ezeken az egyszerűbb zenéken kimagaslónan jól teljesített, legfeljebb 1-1 hangot tippelt mellé, és a hangerő

pontossága is 99% fölötti. Ráadásul a 3 hang, amit az Ancient Nemract zenéjénél nem ismert fel, egy egyszerű programhiba miatt volt a zene legvégén, amit időközben javítottam.

5.2.5 Tesztelés a teljes adathalmazon

Ezeknek az információknak a birtokában már megvannak az elegendően jó paraméterek, hogy a teljes adathalmazon lefuttassam a felismerést. A felismerést Steepest descent-tel, 80-as ismétlészzámmal, az alapdob kihagyásával, és a 0.15 hangerő alatti hangok eldobásával futtattam. A 224 zene konverziója majdnem 3 napig tartott egy 13. generációs, asztali Intel Core i5-ös processzoron, de a fentebbi precizitás elérése érdekében ez elviselhető. Természetesen az optimalizációs módszert biztosan lehetne sokszorosra gyorsítani úgy, hogy ne menjen a felismerés rovására. Sajnos az előző bekezdésben említett programhibára ezután jöttem rá, így szükséges volt egy második adagot futtatni, azoknak a zenéknek a felismerésére, amiket túlzottan érintett; viszont ezalatt más, apró módosítások is történtek (például kijavítottam a gradiensben egy hiányzó 2-es szorzót), így nem teljesen homogén az adathalmaz. A fő paraméterek viszont nem változtak.

A felismerés eredménye a 8-adik fejezetben, a függeléken található, itt összefoglalom az összesített eredményeket. Összességében azt lehet mondani, hogy a 98.40%-os, fentebb definiált hangerő-pontosság nagyon jó, és azt jelenti, hogy a vétett hibák túlnyomó részénél kis különbség van csak az elvárt, és a tényleges hangerő között. A bonyolultabb zenéknél a felismerésben sok felesleges halk hang szerepel, ennek köszönhető a 119846 db olyan hang, ami a referencia zenében nincs ott. Emiatt a felismert hangoknak csak a 83.61%-a található meg az eredeti fájlban (precizitás). Viszont az eredeti zenében szóló hangok 96.80%-át eltalálta az algoritmus, ami kifejezetten jó. A nem eltalált hangok jó része 10%-os, emiatt az algoritmus, ha fel is ismerte, eldobta, mivel 15% alatt volt. A továbbiakban megpróbálom az iterációsztámot növelni, az feltehetően segíteni fog a bonyolultabb zenéknél az eredetiben nem szereplő hangok hangerejének csökkentésében, és így a 15%-os küszöböt is lejjebb lehetne csökkenteni. Ezzel a módszerrel, illetve iterációsztámmal viszont lehet, hogy pontosabb eredményt el lehetne érni 20, vagy 25%-os küszöbértékkal, mivel több a csend hangként felismerése, mint a hang csendként felismerése.

A legrosszabb hangerőhibája a [114] A-U-T-O-M-A-T-I-O-N (Fallen Factory) zenének volt, 82,16%. Ez már a hangfájl exportálásakor túl volt vezérelve a 20dB-es halkítás ellenére, és a felismerendő hangfájl emiatt halkabb volt, mint kellett volna lennie. A hiba nagy

része innen származik. A következő legrosszabb érték a 94.50%, és a [167] Burning Encounter (Boss Battle 16)-hoz tartozik, ezt még részletesebben megvizsgálom később. Viszont 14, egyszerűbb zene kerekítve 100.00%-os eredményt ért el, ami azt jelenti, hogy lényegében teljesen megegyezik az eredetivel.

A legrosszabb jól felismert / összes eredeti érték, azaz az arányaiban véve legtöbb nem felismert hang a [151] Lost at Sea (Misadventure on the Sea 2)-höz tartozik, 64.79%-kal (szennitivitás). Ez tipikusan egy egyszerű, 50 másodperces zene, viszont van benne visszhang effekt, ami azt jelenti, hogy szinte ugyanazt a zenét lemásolták 2-szer, eltolva, halkítva. Mindhárom hangszerből, amelyik szerepel benne, található 10%-os layer is, és amennyiben az ezen szereplő hangokat pontosan fel is ismerte az optimalizáció, a 15%-os küszöbérték miatt minden dobásra kerültek. Ez körülbelül meg is felel a 2/3-os felismerésnek. Egyébként a hangerő pontossága 98.33%, mivel a 10%-os hangerő négyzete nagyon pici. Viszont a 223 zenából 96 esetén az algoritmus az összes eredetileg szóló hangot felismerte.

A legtöbb nem létező hangot az algoritmus a [126] Countdown to Corruption (Boss Battle 11) esetén hallucinálta be, a felismert hangok minden össze 35.23%-a volt ténylegesen ott (precizitás). A zenében levő 6906 hangnak viszont a 99.51%-át megtalálta. A zene egyébként hírhedten a Black MIDI⁴⁴ Note Block-os megfelelője, és ugyan nem tartalmaz kiugróan sok különböző hangot, viszont ahhoz képest rövid, illetve ebbe a számba nincs beleszámolva, hogy sokszor ugyanolyan hangszerű-hangmagasságú Note Block 5-6-szor is szerepel ugyanabban az időpillanatban. A 68722-es csendhez viszonyított hibája a legnagyobb az egész halmazban, és a teljes adathalmaz csendhez viszonyított hibájának 7.3%-át teszi ki. Egyébként ez volt a másik zene, ami, ugyan minimálisan, de túl lett vezérelve. A hangerő pontossága ezek ellenére 95.52%, ami azt mondja, hogy a félreismert hangok betippelése alacsony hangerőn történt. A következő legrosszabb jó tipp / összes tipp arány ismét a [167] Burning Encounter (Boss Battle 16)-é, ezen vizsgálom majd az iterációs zárt, a kezdeti érték, illetve az optimalizáló algoritmus változtatásának hatását. 42 zene esetén viszont egyetlen hangként felismert csend sem volt, azaz minden tipp jó tipp volt. Ezt lehetne úgy is fogalmazni, hogy a precizitás 100%-os volt ezen zenék esetén, viszont számonmra sokkal érhetőbb a tényleges, ehhez a feladathoz kapcsolódó jelentést használni a megfogalmazásban.

⁴⁴https://en.wikipedia.org/wiki/Black_MIDI

Mellékes megjegyzés, hogy az 15A jelű zenében csak alapdob van, így az nem szerepel a függelékben a felismert zenék között, illetve statisztikába sem érdemes beleszámolni.

5.2.6 Algoritmus, kezdeti érték, iterációszám

A [167] Burning Encounter (Boss Battle 16) zenén végeztem több felismerést. Egyszer a Steepest descent-tet futtattam 80-as, illetve 120-as iterációszámmal, illetve a Nonlinear conjugate gradient módszert 120-as iterációszámmal. Ezeket mind 0.5-ös kezdeti értékkel indítottam, míg a 120-as iterációszámú Steepest descent-et kipróbáltam 0.0 kezdőhangerővel is. Fontos megemlíteni, hogy ezek az algoritmusok lokális minimumhelyet keresnek, és ez nem feltétlenül egyezik meg a globális minimummal. A megtalált lokális minimum helye sokban függhet a kezdeti értéktől. Ugyanakkor a tesztek azt mutatják, hogy mind 0.0, mind 0.5-ös kezdeti hangerő esetén elég közel kerül az eredmény az elvárthoz, ami globális minimum, de nincs garantálva, hogy az egyetlen.

Táblázat 3: *Burning Encounter* felismerése különböző paraméterekkel

Kezdeti hangerő	0.5	0.5	0.5	0.0
Algoritmus	SD	SD	NCG	SD
Iterációszám	80	120	120	120
Csend hibája	6952.48	6952.48	6952.48	6952.48
Hangerőhiba	382.26	60.53	60.00	65.74
Eredeti hangok száma	4600	4600	4600	4600
Jól felismert hangok száma	4600	4600	4600	4600
Nem felismert hangok száma	0	0	0	0
Csend hangként felismerése	6068	864	859	494
Jól felismert csend	182232	187436	187441	187806

Az első oszlop felel meg a teljes adathalmazon végzett felismerés paramétereinek. Látható, hogy a 120-as iterációszám jelentősen javította mind a hangerőhibát, mind a halakként felismert fantomhangokat, és nem látható, de jelentősen meghosszabbította az amúgy sem rövid felismerési időt. Ha engedélyezzük a debug naplózást, akkor lehet látni, hogy néhányik iterációnál még mindig nem talált minimumpontot, és észrevehetően csökken a legjobb

költség a 120-adik iterációnál, így szerintem az eredmény tovább lenne javítható az iterációszám növelésével. Pontosabban egy komplexebb megállási feltétel lenne a legcélravezetőbb: ha az elmúlt n (pl. 5) iterációban nem javult a legjobb költség $p\%$ -kal (pl. 5%-kal), akkor kellene leállítani az algoritmust. Ilyen lehetőséget viszont sajnos nem találtam a könyvtárban, az implementáció pedig túlnyúlik az időkereten. Ami elérhető, az egy minimum költség, ami alatt megáll, ezt viszont nem lehet fixen meghatározni. A jelenleg elérhető módszer minden alkalommal elmegy 120 iterációig, akkor is, ha a minimumköltség már a 70-edik iteráció óta lényegében változatlan, és ráadásul az algoritmus a minimum elérése után lassul be igazán, amikor már nincs érdemi haszna.

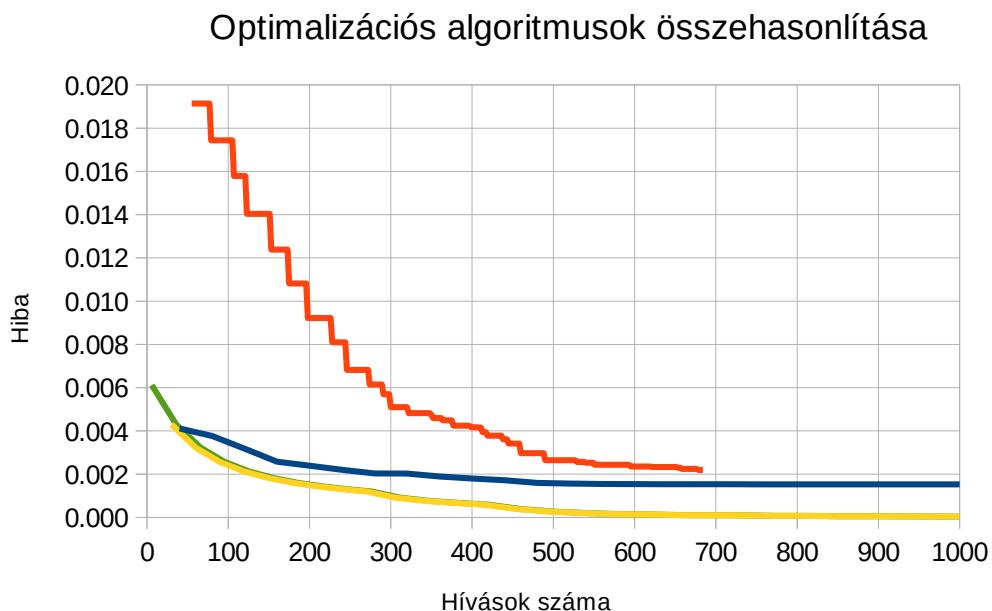
A második és a harmadik oszlopot összehasonlítva lehet látni, hogy az SD és az NCG által adott eredmény között nincs érdemi különbség, illetve a futási idő is körülbelül azonos. Az NCG esetén még egyéb paramétereket is lehet állítani, ezeket az alapértelmezett értékeken hagytam ennél a tesztnél.

Érdekesebb megfigyelni viszont a 0.0 kezdeti érték hatását: ettől sokkal kevesebb lett a csend hangként felismerése, viszont a hangerőhiba nőtt. Ez konzisztens az 1-es táblázatban írtakkal, viszont a mértéke más. Ott a rossz felismerés keveset csökkent, míg a hiba jobban nőtt, itt viszont pont fordítva. Emiatt abban az esetben inkább a 0.5-ös kezdőhangerőt választanám, míg itt lehet, hogy inkább a 0.0-ást. A probléma a 0.0-s indítással az az, hogy nem preferálja a pozitív számokat: a hangfájl szempontjából mindegy, hogy egy negatív szorzót kap, ugyanúgy fog hangzani. A fázisai viszont pont ellentétes fázisban lesznek, ami ezek szerint nem rontja a felismerést túlzottan, viszont mégis jelen információk birtokában észre nem vehető következményei lehetnek. Az exportáláskor a hangerő abszolútértékét veszem figyelembe, így azzal nincsen gond.

Lefuttattam továbbá 7 zenén (044, 047, 057, 060, 099, 110 jelűek) az NCG algoritmust 120-as iterációszámmal, 0.5-ös kezdőhangerővel, és ezeket is össze tudom hasonlítani a függelékben található, 80-as iterációszámú, 0.5-ös kezdőértékű SD futások eredményével. Összességében elmondható, hogy a 120-as iteráció jobban teljesített, kisebb a hangerőhiba (fele-kétharmada), a jól felismert hangok ugyanazok, viszont a hangként felismert csendek száma sokkal kisebb (fele-tizede). Ezt a fejlődést a 3-as táblázatban részletezett teszt miatt az iterációszámnak, és nem az algoritmusváltásnak tulajdonítom.

5.2.7 Algoritmusok grafikonos összehasonlítása

A négy említett algoritmus, a Particle Swarm Optimization, a Nelder-Mead módszer, a Steepest Descent, és a Nonlinear Conjugate Gradient összehasonlítására készítettem egy diagramot, ami a költségfüggvény-hívások számát, és az elért legkisebb hibát ábrázolja. Az algoritmusokat a [035] The Great Race (Great Bridge) első tick-jén futtattam, mivel az kellően bonyolult. 8 hárfa, 3 basszusgitár, és két pergődob van egy időpillanatban (amik közül több duplikált), de az előfelismerés sok klikket is felismer, amikhez 0 hangerőt kell rendelni. Összesen 54 optimalizálandó paraméter található benne. Az utóbbi két módszer a gradienst is kiszámoltatja, és ahhoz, hogy egy diagramra tudjam őket tenni, és a skála nagyjából egyezzen, ezt 5 költségfüggvény-hívásként számítottam be. Persze 54 paraméter esetén ez 54 parciális derivált kiszámolását jelenti, viszont a számolás jó része közös a deriváltak között.



Ábra 9: Optimalizációs algoritmusok összehasonlítása

A Nelder-Mead módszer konvergált, míg a Particle Swarm Optimization nem tudott jobb eredményt elérni. A maradék két algoritmus lényegében egybeesik, és sokkal jobb eredményt ért el, mint az előbbi kettő. A SD és a NCG a hangokat lényegében eltalálták, az egyik pergődobnál tippelek 1.0 helyett 0.9-es hangerőt. A NM hangerőhibája a csendhez viszonyítva több, mint 3-szoros lett, és a 8 jó hang mellett 36 rossz hangot is betippelt, a klikkeket nem sikerült kioptimalizálnia. A PSO hangerőhibája picit lett nagyobb a csendénél,

8 hangból 6-ot ismert fel, és 12 rosszat is betett. Összességében tehát majdhogynem összehasonlíthatatlanul jobb a gradienst felhasználó algoritmusok által adott eredmény.

5.3 Tempófelismerés pontossága

A tempófelismerésen sokat lehetne javítani. Jelenleg az aubio könyvtárat használom, és apró heurisztikával próbálom javítani az eredményt, az onset-ek figyelembevételével. A tps egzakt módon meghatározható, ha feltesszük, hogy 0.25 tps-enként van kvantálva. Az ONBS felületén alapesetben ezeket a tempókat lehet beállítani, viszont ha valaki szeretne, pontos értéket is megadhat. Túlnyomórészt viszont a zenék tps-e a 0.25 többszöröse. A teszthalmazban is követi ezt az összes zene. A leggyakoribb a 10, és a 6.75 tps, mivel ezek osztják a Minecraft (mint program) 20 tps-es belső órajelét.

A 223 zenából 108 tps-t sikerült eltalálni, 5-nek a felét, 4-nek a kétszeresét tippelte a program. 5 esetén 0.25-tel kisebb értéket, 11 esetén 0.25-tel nagyobb értéket talált. A maradék 90 érték nincs értékelhetően közel az elvárthoz. A 108 eltalált érték közül 90 db 6.75, és 15 db 10.00 található. A teljes adathalmazban 140 db 6.75, míg 70 db 10.00 tps-t felhasználó zene van. A többi érték a WynnCraft zenéi között kifejezetten ritka.

Látva a rossz eredményt, kíváncsi lettem, hogy egy másik könyvtár mennyivel tudná jobban eltalálni a tempót. Választásom a DeepRhythm⁴⁵ nevű, neurális háló alapú Python könyvtárra esett, és tisztán teszt célból lefuttattam vele is a felismerést. Az általa kiadott BPM értéket szorzom 4/60-nal, a tps-é történő konverzió érdekében, illetve kvantálom a legközelebbi negyedhez. A felhasznált forráskód, illetve az eredmények ugyanott⁴⁶ megtalálhatóak.

Az eredmény valóban jóval jobb lett, ugyanazokon a fájlokon 185 helyes eredményt adott (amiből 126 db 6.75, 54 db 10.00, és 5 egyéb), 3-szor konzisztensen 6 tps-t tippelt a 12 helyett; és kétszerest, illetve 0.25-tel eltoltat nem tippelt. 15 helyen 6.5-öt ismert fel 10 helyett (kétharmada), illetve 12 helyen 9 tps-t talált a 6.75 helyett (háromnegyede). Valószínűleg ezeknek részben az az oka, hogy én 4-gyel szoroztam a beat-ból tick-be történő konverziókor, viszont léteznek hármas lüktetésű zenék is. A maradék 8 esetből 3-nál crash-elt a program, túl rövid lehetett a zene, a többi aránya szintén közel van kis számok arányához.

⁴⁵<https://pypi.org/project/deeprhythm/>

⁴⁶https://github.com/4321ba/mp3-to-nbs/tree/main/WynnCraft_Noteblock_OST

A könyvtár integrálásával olyan gyakorlati problémám van, hogy Rust kódból nem tűnik igényes megoldásnak Python könyvtárat hívni.

5.4 Összehasonlítás eddig elérhető módszerekkel

A kutatásaim szerint eddig elérhető automatizált módszer két lépésből áll: egy MIDI fájl generálása az egyik (irodalomkutatás során említett) hullámos hangfájlból MIDI készítő programmal, majd Note Block Studio-ba a MIDI importálása. Ezzel a módszerrel rögtön elveszítjük a hangszer-információkat, illetve, amennyiben ütős hangszereket is megpróbálunk felismerni, kétlem, hogy a hangmagasság-információt bárhogyan meg lehetne tippelni.

Ezek figyelembenélővel első tesztként a [157] Simplified What Must Be Done (The Sacrifice) című dalt tesztelem elsőként, mivel ez a lehető legegyszerűbb, és csak hárfa van benne. A simplified jelző azt jelenti, hogy a halkabb hangokat eltávolítottam belőle, így a visszhang effekt sem okozhat problémát. A felismeréshez a 2.2.3-ban bemutatott Basic Pitch nevű konvertáló eszközt használom. Ez az egyszerű dallamot kimondottan jól felismerte. A Note Block Studio-ba történő importálás is egyszerűen ment, és közeli eredményt adott az eredetihez. Probléma itt is van a tempóval, de a Basic Pitch oldalán lehetőség nyílt az exportált MIDI tempóját beállítani, amit a megfelelő értékre állítva a Note Block Studio-ba importáláskor is jó tempójú lett a zene. Az összehasonlító szkriptem az eredeti fájllal nem tudja jól összehasonlítani (a 223-ból mindenben 14 hangot talál megfelelőnek), mivel van a fájlból némi időbeli pontatlanság, elcsúszás, és emiatt nem ugyanabban a tick-ben vannak az összehasonlítandó hangok. Egy kis manuális közbeavatkozással meg lehet oldani, hogy a 223 hangból 221-et eltaláljon. Nem létező hangból is csak 20-at tett be. Ezek nagyon jó arányok, de ismét fontos megjegyezni, hogy ez a lehető legegyszerűbb szám. Ez az egyik szám, amit a módszerem tökéletesen eltalált.

Másik próbálkozásként az egyik bonyolultabb zenét, az 5.2.6-ban is tárgyalt [167] Burning Encounter (Boss Battle 16)-t próbálom felismerni. Ez 12 tps-es, így 180-as bpm-et kell beállítsak a Basic Pitch oldalán. Mivel a Basic Pitch nem ismer fel ütős hangszert, így a 4600-ból 1643 hang felismerésére nincsen módunk. A maradék 2957 hang pedig két külön hangszeren van, amit a Basic Pitch (sem általában más, hasonló szoftver) nem ismer fel. Hangmagasság alapján szét lehetne öket választani, és az ONBS importáló felületén van is olyan kapcsoló, hogy a 2 oktávos skálán kívül eső hangokat ne konvertálja belülre, viszont a hangok egyszerű szétválasztására nincs mód. ONBS-ben legalábbis. A felhasználó meg tud

nyitni egy MIDI szerkesztő⁴⁷, és külön hangszer sávjára tenni a megfelelő hangokat. Ezután az ONBS-ben megnyitva lehet a hangszereket megfelelően leképezni. Az így kapott NBS fájl visszajátszva nagyjából felismerhető, hogy melyik eredeti zene átirata, de nagyon távol áll attól, hogy jónak mondhattam. Viszont szerencsére az ütem nem csúszott el, így használható az összehasonlító szkript.

Itt érdemes referenciának venni a 3-as táblázatot. A hangerőhiba nem bír túlzottan nagy jelentőséggel itt, legfeljebb két MIDI konverzió között, mivel konstanssal szorzás sokat módosíthat rajta. Érdekesebbek lehetnek viszont az eltalált hangok darabszámai. Az összehasonlításhoz nem veszem figyelembe az ütős hangszereket, csak a hárfát és a basszusgitárt. Két eredményt teszek bele a táblázatba, az egyiknél csak a hárfát veszem figyelembe a MIDI-ból, a másiknál minden hangszeret. Referenciaként bemásolom a 0.0 kezdetű, 120 iterációs Steepest descent futtatás eredményét is, az ütős hangszerek figyelembe vétele nélkül.

Táblázat 4: MIDI-ből történő konverzió értékelése bonyolult zenén

	Forrás	SD 120	MIDI	MIDI
Hangszerek	H, B	H	H, B	
Csend hibája	5309.48	5309.48	5309.48	
Hangerőhiba	55.05	3613.61	3424.23	
Eredeti hangok száma	2957	2957	2957	
Jól felismert hangok száma	2957	1087	1550	
Nem felismert hangok száma	0	1870	1407	
Csend hangként felismerése	494	342	934	
Jól felismert csend	92999	93151	92559	

A hárfákat nem túl rosszul felismerte, 1087-et eltalált, és 342-t tett rossz helyre. A basszusgitárról ez már nem mondható el: a harmadik és a második oszlop különbsége mutatja, hogy 463 basszusgitár került megfelelő helyre, és 592 nem. Ráadásul van még összesen 1407 hang, amit nem talált meg, viszont a hangok több, mint felét megtalálta. Ezzel szemben a szakdolgozatban taglalt módszer az összes hangot megtalálta, és ugyan a 494 hang, amit

⁴⁷<https://signal.vercel.app/>

hamisan ismert fel, sok, viszont a hangerőhibából látszik, hogy ezek halkak, így nem olyan zavaróak.

Látszik a probléma a MIDI-n kereszttüli módszerrel: általában kellenek manuális módosítások, és az eredmény azokkal sem lesz jó. Emiatt tapasztalatom szerint gyakoribb a hallás utáni, teljesen manuális átírás.

5.5 Robusztusság

A robusztusság fontos kérdés a való életbeli használathoz. Sajnos ez az algoritmus, mivel az nbswave használatát feltételezi, elég alacsony mértékben használható más forrásokból származó Note Block-os zenék felismerésére, ide tartozik a játékból, vagy a Stúdióból felvett visszajátszás, a régebbi exportőrből származó zenék, továbbá a YouTube, illetve egyéb tartalomszolgáltató által használt agresszív tömörítőeljárások is ronthatnak a felismerés minőségén. Továbbá ha a zene kezdőpillanata nem a 0 indexű mintánál van, hanem van egy pici csend az elején, az külön kezelést igényel: meg kell határozni a tényleges kezdőpillanatot, és onnantól kell a felismerést indítani.

6 Összefoglaló

Összességében megismerkedtem az automata zeneátírás (AMT) nehézségeivel, és a Note Block-os zenék sajátosságaival. Feltérképeztem a témaban eddig elérhető módszereket, programokat, és megpróbáltam velük átírni Note Block-os zenéket hullámformából kottaszűrű reprezentációba. Készítettem egy parancssori programot, ami az eddig elérhetőnél sokkal jobb eredményt nyújt az nbswave által generált hangfájlok visszaírására. Végül sokoldalúan teszteltem a programot, különféle paraméterek változtatásával több zenén, és egy 9.5 órás adathalmazon is lefuttattam a felismerést, ami több, mint 600 000 darab hangot tartalmazott.

A jövőben a futásidőn, illetve a különböző forrásokból származó Note Block-os zenék átírásának pontosságán, azaz a robusztusságon lehetne sokat javítani. Ehhez tartoznak az egyenetlenül eloszló onset-ek, esetleges túlvezérlés, illetve a túlvezérlés-kompenzáció, hangerőcsökkentés felismerése, illetve esetlegesen a régebbi exportör hangmagasság-pontatlansága. Hasznos volna további hangszerek felismerését tesztelni, illetve az alapdob felismerésén javítani. A több hangszer teszteléséhez egy másik adathalmazon is célszerű volna a konverziót elvégezni, ami többféle, újabb hangszeret is tartalmaz. Ezenkívül lehetne a WynnCraft-os adathalmazt a saját hangmintáival is felismerni, és megvizsgálni, hogy annak felhasználásával hogyan alakul a felismerés pontossága. Javítani szükséges továbbá a tempófelismerésen, esetleg egy másik könyvtár felhasználásával. Érdemes lehet megvizsgálni azt is, hogy mennyit segít a felismerésen, ha a kezdeti hangerőt hangonként eltérő értékre állítjuk be, például az előfelismerés által adott eredményt felhasználva. Az eredeti hangfájlban leggyakrabban a layerek hangerejeit állítják, így hasznos lehet az azonos hangszerek hangjait egy-egy különálló, megegyező hangerejű halmazba csoportosítani, így apró hangerőbeli pontatlanságok is felismerhetők volnának.

Az elért eredmény viszont, a 98.4%-os hangerőpontosság a teljes adathalmazon nagyon biztosító, és mutatja, hogy a formátum sajátosságait kihasználva tényleg sokkal nagyobb pontosság érhető el, mint az általános módszereket alkalmazva.

7 Irodalomjegyzék

- 1: Brilliant Classics, 100 Best of Beethoven, Hozzáférés dátuma: 2024. 09. 30.,
<https://youtu.be/Wo-hwx-TWUw?feature=shared>
- 2: ClassicMan, Beethoven: Für Elise, Hozzáférés dátuma: 2024. 09. 30.,
<https://musescore.com/classicman/fur-elise>
- 3: MIDI Association, MIDI specifications, Hozzáférés dátuma: 2024. 09. 30.,
<https://midi.org/specs>
- 4: IBM; Microsoft, Multimedia Programming Interface and Data Specifications 1.0, August 1991, <https://www.aelius.com/njh/wavemetatools/doc/riffmci.pdf>
- 5: ISO/IEC 13818-3:1998, Information technology — Generic coding of moving pictures and associated audio information - Part 3: Audio, 1998, <https://www.iso.org/standard/26797.html>
- 6: Recordare, MakeMusic, MusicXML Publications, 2024. 09. 30.,
<https://www.musicxml.com/publications/>
- 7: Wikipedia, List of best-selling video games, Hozzáférés dátuma: 2024. 09. 30.,
https://en.wikipedia.org/wiki/List_of_best-selling_video_games
- 8: Minemanó, Graphing Calculator Without Commands 99×99 Pixel (Redstone) [Minecraft], Hozzáférés dátuma: 2024. 09. 30., <https://youtu.be/qd5khvoO4tI?feature=shared>
- 9: Minecraft Wiki, Hangdobj, Hozzáférés dátuma: 2024. 09. 30.,
<https://minecraft.fandom.com/hu/wiki/Hangdobj>
- 10: OpenNBS contributors, Open Note Block Studio, Hozzáférés dátuma: 2024. 09. 30.,
<https://opennbs.org/>
- 11: OpenNBS, NBS Format, Hozzáférés dátuma: 2024. 09. 30., <https://opennbs.org/nbs>
- 12: Minecraft Server List, Most Players Online Servers, Hozzáférés dátuma: 2024. 09. 30.,
<https://minecraft-server-list.com/sort/PlayersOnline/>
- 13: , Hypixel Website, Hozzáférés dátuma: 2024. 11. 15., <https://hypixel.net/>
- 14: WynnCraft fejlesztők, WynnCraft, Hozzáférés dátuma: 2024. 09. 30.,
<https://wynncraft.com/>

- 15: ShinkoNet, Hypixel Skyblock OST, Hozzáférés dátuma: 2024. 09. 30.,
<https://www.youtube.com/playlist?list=PLPYaA8L35a72GLLbbMKc2v8D-AHPDFXsV>
- 16: WynnCraft composers, WynnCraft OST, Hozzáférés dátuma: 2024. 09. 30.,
<https://www.youtube.com/playlist?list=PLXDDjHAEqnMoKKzMwoKLtENh-XM601H>
- 17: D. Abrahams, ENGAGING MUSIC STUDENTS THROUGH MINECRAFT, 2018,
<https://library.iated.org/view/ABRAHAMS2018ENG>
- 18: 윤관기, A Study on the Teaching Method of Melody Creation Using Minecraft Note Block, 2022.11, <https://www.dbpia.co.kr/Journal/articleDetail?nodeId=NODE11250375>
- 19: E. Benetos, S. Dixon, Z. Duan and S. Ewert, "Automatic Music Transcription: An Overview," in IEEE Signal Processing Magazine, vol. 36, no. 1, pp. 20-30, Jan. 2019,
<https://ieeexplore.ieee.org/abstract/document/8588423/metrics#metrics>
- 20: Liu, L., Benetos, E., From Audio to Music Notation. In: Miranda, E.R. (eds) Handbook of Artificial Intelligence for Music. Springer, Cham, 2021, https://doi.org/10.1007/978-3-030-72116-9_24
- 21: Maman, Ben and Bermano, Amit H, Unaligned Supervision for Automatic Music Transcription in The Wild, 2022, <https://proceedings.mlr.press/v162/maman22a.html>
- 22: Yu-Te Wu, Yin-Jyun Luo, Tsung-Ping Chen, I-Chieh Wei, Jui-Yang Hsu, Yi-Chin Chuang, Li Su, Omnidart: A General Toolbox for Automatic Music Transcription, 2021, <https://doi.org/10.48550/arXiv.2106.00497>
- 23: Frigiduck, Eight-Legged Foe, Hozzáférés dátuma: 2024.11.25.,
<https://www.youtube.com/watch?v=rfhYBA7xQsI>
- 24: McLeod, Andrew & Schramm, Rodrigo & Steedman, Mark & Benetos, Emmanouil, Automatic Transcription of Polyphonic Vocal Music. Applied Sciences. 7. 1285. 10.3390/app7121285, 2017,
https://www.researchgate.net/publication/321734596_Automatic_Transcription_of_Polyphonic_Vocal_Music
- 25: W. U. D. Rodrigo, H. U. W. Ratnayake, and I. A. Premaratne, Identification of Music Instruments from a Music Audio File, 2021, https://doi.org/10.1007/978-981-33-4355-9_26
- 26: H. Li, H. You, X. Fei, M. Yang, K. -M. Chao and C. He, "Automatic Note Recognition and Generation of MDL and MML using FFT," 2018 IEEE 15th International Conference on

e-Business Engineering (ICEBE), Xi'an, China, 2018, pp. 195-200, 2018,
<https://doi.org/10.1109/ICEBE.2018.00038>

27: Daniel Lee, H. Sebastian Seung, Algorithms for Non-negative Matrix Factorization, 2000,
https://proceedings.neurips.cc/paper_files/paper/2000/hash/f9d1152547c0bde01830b7e8bd6024c-Abstract.html

28: Rachel M. Bittner and Juan José Bosch and David Rubinstein and Gabriel Meseguer-Brocal and Sebastian Ewert, A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation, 2022, <https://arxiv.org/abs/2203.09893>

29: Wikipedia, Equal temperament, Hozzáérés dátuma: 2024. 11. 01.,
https://en.wikipedia.org/wiki/Equal_temperament

30: Jorge Nocedal and Stephen J. Wright, Numerical Optimization, 2006,
<https://shuyuej.com/books/Numerical%20Optimization.pdf>

8 Függelék

A teljes adathalmazon végzett felismerés eredménye itt található.

Fájlnév	Csend hibája	Hangerő hiba / csend hibája)	Hangerő pontos- sága (1 – hangok száma	Eredeti hangok száma	Jól felis- mert	Nem felis- mert	Jól felis- mert / összes eredeti	Csend hang- ként felis- merése	Jól felis- mert / összes tipp	Jól felis- mert
001_Luxury_of_the_Cease-Fire_Ragni.nbs		722,56	0,01	100,00%	1039	1039	0	100,00%	0	100,00%
002_Sheltered_Settlement_Ravine_Village.nbs		3 350,51	124,71	96,28%	3333	3214	119	96,43%	131	96,08%
003_The_Binding_Alliance_Maltic.nbs		3 039,00	2,00	99,93%	1561	1561	0	100,00%	17	98,92%
004_Triumphant_Citadel_Deltas.nbs		1 798,00	2,04	99,89%	1696	1696	0	100,00%	17	99,01%
005_Understated_Pride_Elkurn.nbs		3 798,38	32,69	99,14%	2162	2144	18	99,17%	124	94,53%
006_Muddy_Outlook_Nemract.nbs		1 389,00	2,80	99,80%	1151	1151	0	100,00%	28	97,63%
007_Saddle_Up_Ternaves.nbs		3 020,00	0,30	99,99%	1344	1344	0	100,00%	0	100,00%
008_Sacred_Hospitality_Bremminglar.nbs		1 502,00	0,63	99,96%	830	830	0	100,00%	0	100,00%
009_Jewel_of_the_Desert_Almuj.nbs		3 307,00	17,95	99,46%	1495	1493	2	99,87%	129	92,05%
010_Shady_Deals_Abounding_Rymek.nbs		1 125,00	0,18	99,98%	1119	1119	0	100,00%	2	99,82%
011_Snowdrifts_Billowing_Nesaak.nbs		3 116,00	0,16	99,99%	1870	1870	0	100,00%	0	100,00%
012_Frozen_Waltz_Lusuco.nbs		2 029,12	0,01	100,00%	1866	1866	0	100,00%	0	100,00%
013_Harmonic_Tradition_Iboju_Village.nbs		4 363,60	56,18	98,71%	2406	2406	0	100,00%	627	79,33%
014_Wynns_Shining_Fortress_Troms.nbs		4 406,00	0,74	99,98%	2264	2264	0	100,00%	3	99,87%
015_Epic_Wynn_Ragni_Outskirts.nbs		4 797,00	21,58	99,55%	2187	2187	0	100,00%	241	90,07%
016_Adventure_Emerald_Trail.nbs		2 064,00	5,75	99,72%	2040	2040	0	100,00%	29	98,60%
016_Modified_Adventure_Emerald_Trail.nbs		2 675,00	8,01	99,70%	1440	1440	0	100,00%	72	95,24%
017_Forest_Dance_Nivla_Woods.nbs		2 400,18	6,52	99,73%	2624	2620	4	99,85%	14	99,47%
017_Modified_Forest_Dance_Nivla_Woods.nbs		2 066,09	3,12	99,85%	1383	1381	2	99,86%	7	99,50%
018_Warrior_Adventure_Maltic_Plains.nbs		3 924,00	6,48	99,83%	3128	3124	4	99,87%	0	100,00%
019_Fresh_Breeze_of_Salt_The_Coast.nbs		7 198,32	13,17	99,82%	3810	3810	0	100,00%	205	94,89%
020_Treacherous_Walkways_Pigmans_Ravines.nbs		3 814,00	0,33	99,99%	2054	2054	0	100,00%	0	100,00%
021_In_The_Little_Wood_Little_Wood.nbs		4 448,00	2,22	99,95%	2366	2364	2	99,92%	1	99,96%
022_Remnants_of_the_Past_Time_Valley.nbs		2 190,64	3,86	99,82%	2385	2378	7	99,71%	37	98,47%
023_Quiet_Beginnings_Deltas_Suburbs.nbs		880,00	0,00	100,00%	654	654	0	100,00%	0	100,00%
024_Unfamiliar_Lands_Nemract_Swamp.nbs		3 351,00	10,85	99,68%	2357	2356	1	99,96%	124	95,00%
025_Defiled_Sanctity_Graveyard.nbs		1 824,00	0,01	100,00%	820	820	0	100,00%	0	100,00%
026_Yearning_for_the_Days_of_Glory_Ancient_Nemract.nbs		1 357,60	3,67	99,73%	1372	1369	3	99,78%	2	99,85%
027_Black_Road_Nostalgia_Black_Road.nbs		845,00	0,28	99,97%	839	839	0	100,00%	0	100,00%
027_Modified_Black_Road_Nostalgia_Black_Road.nbs		831,00	2,32	99,72%	825	823	2	99,76%	0	100,00%
028_Ballad_of_the_Trees_Savanna.nbs		1 906,26	2,14	99,89%	1641	1641	1	99,94%	26	98,44%
029_Travelling_at_Dusk_Desert.nbs		4 436,00	1,56	99,96%	2556	2556	0	100,00%	2	99,92%
030_No_Mans_Land_Roots_of_Corruption.nbs		3 096,96	22,21	99,28%	3042	3000	42	98,62%	284	91,35%
031_Mining_Rhythms_Abandoned_Mines.nbs		1 521,40	10,98	99,28%	2096	2013	83	96,04%	86	95,90%
032_Sun-Stained_Stones_Mess_Canyons.nbs		2 404,00	0,62	99,97%	1440	1440	0	100,00%	2	99,86%
033_Icy_Steps_Frozen_Forest.nbs		2 436,00	5,13	99,79%	1538	1538	0	100,00%	112	93,21%
034_Frozen_Lands_Ice_Canyon.nbs		1 560,00	0,11	99,99%	972	972	0	100,00%	0	100,00%
035_The_Great_Race_Great_Bridge.nbs		9 290,00	177,79	98,09%	4032	3992	40	99,01%	1685	70,32%
036_The_Legend_Begins_Troms_Jungle.nbs		1 101,00	3,81	99,65%	1101	1101	0	100,00%	2	99,82%
037_Uncivilization_Dernel_Ruins.nbs		4 825,00	23,40	99,52%	2545	2540	5	99,80%	239	91,40%
038_Dormancy_Reversal_Mount_Wynn.nbs		4 988,60	40,00	99,20%	2724	2713	11	99,60%	664	80,34%
039_Bumps_in_the_Night_Hallowynn_2014.nbs		1 028,00	0,00	100,00%	974	974	0	100,00%	0	100,00%
03A_Corkian_Madness_Dwelling_Walls.nbs		8 349,92	92,59	98,89%	3389	3319	260	92,33%	342	90,15%
040_Accursed_Dunes_Mummys_Tomb.nbs		803,00	0,00	100,00%	782	782	0	100,00%	0	100,00%
041_Allegiances_as_Quicksand_Takanos_Barracks.nbs		613,20	0,32	99,95%	657	657	0	100,00%	0	100,00%
042_Mission_Wynnpossible_Rymek_Mansion.nbs		4 518,74	45,55	98,99%	3398	3382	16	99,53%	879	79,37%
043_Visions_of_the_Warrior_Bobs_Tomb.nbs		690,02	0,98	99,86%	750	714	36	95,20%	2	99,72%
044_Echoes_of_a_Bygone_Era_Time_Travel.nbs		1 560,64	45,85	97,06%	3830	3060	770	79,90%	313	90,72%
045_War_Weary_Reverie_Past_Nesaak.nbs		4 068,90	7,14	99,82%	3326	3314	12	99,64%	56	98,34%
046_Forlorn_Halls_House_of_Twain.nbs		5 025,58	20,58	99,59%	3854	3253	601	84,41%	30	99,09%
047_Trespassers_will_be_Excised_Emerald_Labyrinth.nbs		3 816,17	12,61	99,67%	2277	2275	2	99,91%	165	93,24%
048_Staggering_March_Ariodos_Lab.nbs		2 397,43	18,17	99,24%	2102	2062	40	98,10%	43	97,96%
049_Spirited_Sailing_Stories_Selchar.nbs		4 039,00	17,48	99,57%	1821	1819	2	99,89%	70	96,29%
050_Antics_of_the_Fantastical_Mage_Island.nbs		2 291,00	2,85	99,88%	1245	1245	0	100,00%	46	96,44%
051_The_Unforgiving_Slopes_Ice_Islands.nbs		3 376,00	10,68	99,68%	1784	1782	2	99,89%	14	99,22%
052_At_the_End_of_the_Alley_Pirate_Cove.nbs		5 610,96	19,46	99,65%	3798	3798	0	100,00%	402	90,43%
053_High_Tides_Ocean_1.nbs		2 602,00	8,60	99,67%	1365	1365	0	100,00%	162	89,39%
053_Modified_High_Tides_Ocean_1.nbs		2 602,00	17,53	99,33%	1365	1360	5	99,63%	178	88,43%
054_Stormy_Seas_Ocean_2.nbs		3 527,00	1,17	99,97%	2041	2041	0	100,00%	4	99,80%
055_Balada_del_Vagabundo_Marino_Ocean_3.nbs		1 892,30	10,57	99,44%	2288	2262	26	98,86%	70	97,00%
056_Pioneered_Triumph_Volcanic_Isles.nbs		6 577,66	51,80	99,21%	4249	4245	4	99,91%	663	86,49%
057_The_Stronghold_Yet_Stands_Skiens_Island.nbs		2 838,31	20,63	99,27%	3901	3887	14	99,64%	284	93,19%
058_Forgotten_Excalibur_Maro_Peaks.nbs		3 330,32	15,16	99,54%	1895	1895	0	100,00%	344	84,64%
059_Machinations_of_Chaos_Corkus.nbs		2 609,84	9,53	99,63%	2192	2192	0	100,00%	151	93,56%
060_Wings_of_Distrust_Avos_Theme.nbs		3 869,08	8,51	99,78%	2907	2907	0	100,00%	137	95,50%
061_Settlements_of_Freedom_Corkus_Docks.nbs		3 632,20	86,84	97,61%	3630	2975	655	81,96%	227	92,91%
061_Simplified_Settlements_of_Freedom_Corkus_Docks.nbs		3 384,56	65,76	98,06%	2176	2123	53	97,56%	233	90,11%
062_Corkian_National_Anthem_Corkus_City.nbs		5 545,14	43,96	99,21%	3190	3168	22	99,31%	321	90,80%
063_Waters_of_Prosperity_Relos.nbs		3 355,36	9,95	99,70%	3135	3119	16	99,49%	61	98,08%
064_Craftmas_Carol_Craftmas_Island.nbs		1 030,62	3,52	99,66%	1878	1864	14	99,25%	0	100,00%
065_Antiquated_Ancestry_Dead_Island.nbs		1 520,04	0,24	99,98%	1781	1765	16	99,10%	0	100,00%
066_Monument_of_a_New_Land_Llevigar.nbs		4 773,00	13,57	99,72%	2491	2491	0	100,00%	267	90,32%
067_Humble_Beginnings_Bucie.nbs		2 091,46	0,90	99,96%	2481	2481	0	100,00%	0	100,00%
068_Gray_Skies_Dark_Days_Olux.nbs		7 136,00	2,58	99,96%	3250	3248	2	99,94%	0	100,00%
069_Lament_of_the_Decay_Gelibord.nbs		631,25	0,94	99,85%	1391	1391	0	100,00%	0	100,00%
070_Camaraderie_Efilim.nbs		1 584,12	14,24	99,10%	1353	1353	0	100,00%	196	87,35%
071_No_Place_to_Hide_Lexdale.nbs		5 267,62	52,85	99,00%	4262	4120	142	96,67%	524	88,72%
071_Simplified_No_Place_to_Hide_Lexdale.nbs		4 027,18	17,68	99,56%	2702	2691	11	99,59%	216	92,57%
072_Shining_Citadel_of_the_Allies_Cinfras.nbs		4 318,00	10,58	99,75%	2346	2346	0	100,00%	164	93,47%
073_To_The_Sky_Letus_Airbase.nbs		3 533,00	25,69	99,27%	1935	1918	17	99,12%	164	92,12%
074_Horizon_of_the_East_Aldorei_Valley.nbs		2 533,00	0,68	99,97%	1027	1027	0	100,00%	4	99,61%

Fájlnév	Csend hibája	Hangerő pontos- sága (1 – hiba / csend hibája)	Eredeti hangok száma	Jól felis- mert hangok száma	Nem felis- mert hangok száma	Jól felis- mert / összes eredeti	Csend hang- ként felis- mertére	Jól felis- mert / összes tipp	Jól felis- mert csend
075_The_Hardest_of_Souls_Thanos.nbs	1 794,00	20,55	98,85%	1716	1702	14	99,18%	14	99,18%
076_Anthem_of_the_Air_Mage_Bantis_Air_Temple.nbs	4 079,24	64,17	98,43%	3906	3835	71	98,18%	380	90,98%
077_Hard_Work_Alone_Thesead.nbs	4 194,00	3,75	99,91%	2068	2068	0	100,00%	55	97,41%
078_Fine_Work_Together_Eltom.nbs	2 215,80	21,87	99,01%	1843	1831	12	99,35%	122	93,75%
079_Pride_Progress_and_Digging_Holes_Rodoroc.nbs	3 116,00	34,15	98,90%	1478	1478	0	100,00%	582	71,75%
080_Hideaway_of_the_Stalwart_Maex.nbs	2 586,08	6,49	99,75%	2048	2048	0	100,00%	97	95,48%
081_Standing_on_Thin_Air_Kandon-Beda.nbs	3 247,00	0,73	99,98%	1527	1527	0	100,00%	6	99,61%
082_City_in_the_Sky_Ahmsord.nbs	3 233,20	6,85	99,79%	2185	2183	2	99,91%	49	97,80%
083_Shanty_of_the_Somber_Sailors_Jofash_Docks.nbs	2 591,40	1,46	99,94%	3465	3465	0	100,00%	17	99,51%
084_Epic_Gavel_Gavel_Gate.nbs	4 264,00	1,84	99,96%	2224	2224	0	100,00%	8	99,64%
085_Gavel_Journey_Llevigar_Plains.nbs	4 458,00	9,15	99,79%	2424	2422	2	99,92%	97	96,15%
085_Modified_Gavel_Journey_Llevigar_Plains.nbs	4 374,00	11,73	99,73%	2392	2388	4	99,83%	109	95,63%
086_Ten_Minutes_til_Breaktime_Karoc_Quarry.nbs	2 186,92	1,02	99,95%	2085	2085	0	100,00%	0	100,00%
087_Trodden-Down_Paths_Olux_Swamp.nbs	1 135,00	0,00	100,00%	1135	1135	0	100,00%	0	100,00%
088_The_Forest_Lives_On_Light_Forest.nbs	4 059,00	8,22	99,80%	2101	2101	0	100,00%	78	96,42%
089_The_Forest_Dies_Dark_Forest.nbs	1 782,00	3,55	99,80%	1728	1728	0	100,00%	40	97,74%
090_The_Outlook_is_Bleak_Kander_Woods.nbs	3 115,92	4,74	99,85%	2210	2178	32	98,55%	43	98,06%
091_Gylia_Shanty_Cinfras_County.nbs	3 775,00	0,01	100,00%	2019	2019	0	100,00%	0	100,00%
092_Alimless_Paths_Canyon_of_the_Lost.nbs	2 705,00	4,77	99,82%	1225	1221	4	99,67%	1	99,92%
093_Clear_Day_Shadows_from_Above_Ozoths_Spire.nbs	4 166,84	7,52	99,82%	1979	1979	0	100,00%	102	95,10%
094_Call_of_the_Orient_Crystal_Falls.nbs	5 647,96	30,41	99,46%	4058	4056	2	99,95%	483	89,36%
095_Blazing_Mountain_Molten_Caverns.nbs	3 111,00	6,82	99,78%	1553	1553	0	100,00%	134	92,06%
096_Peaks_of_Ash_and_Flame_Molten_Heights.nbs	1 300,16	11,66	99,10%	1576	1569	7	99,56%	45	97,21%
097_Amongst_the_Clouds_Sky_Islands.nbs	4 298,00	58,09	98,65%	2481	2481	0	100,00%	839	74,73%
098_Dance_of_Memory_Caritat_Manor.nbs	768,88	0,00	100,00%	844	844	0	100,00%	0	100,00%
099_Glorious_Illusions_Trippy_Forest.nbs	2 664,28	15,04	99,44%	2257	2257	0	100,00%	191	92,20%
100_Lights_Hymn_Light_Realm.nbs	4 307,00	2,46	99,94%	1443	1441	2	99,86%	4	99,72%
101_Simplified_Tale_of_the_Century_Flight_in_Distress.nbs	7 932,48	116,26	98,53%	4716	4702	14	99,70%	1548	75,23%
101_Tale_of_the_Century_Flight_in_Distress.nbs	10 753,12	190,75	98,23%	7730	7199	531	93,13%	2315	75,67%
102_Burning_Soul_Fleris_Cave.nbs	6 147,12	77,30	98,74%	3528	3528	0	100,00%	1423	71,26%
103_Into_the_Unknown_Unknown.nbs	4 795,12	6,47	99,87%	2270	2268	2	99,91%	66	97,17%
104_Stumbling_to_the_Brink_of_Despair_The_Void.nbs	1 345,36	2,35	99,83%	1282	1282	0	100,00%	39	97,05%
105_Clear_Sky_Lullaby_World_Above_the_Clouds.nbs	3 525,70	58,83	98,33%	5388	4360	1028	80,92%	19	99,57%
105_Simplified_Clear_Sky_Lullaby_World_Above_the_Clouds.nbs	1 686,16	0,56	99,97%	1634	1634	0	100,00%	0	100,00%
106_Anthem_of_Rot_Decrepit_Sewers.nbs	2 649,50	20,31	99,23%	2521	2507	14	99,44%	31	98,78%
107_Arachnophobia_Infested_Pit.nbs	2 041,00	2,10	99,90%	925	925	0	100,00%	38	96,05%
108_Bottoms_of_the_Food_Chain_Lost_Sanctuary.nbs	3 494,36	20,93	99,40%	2590	2451	139	94,63%	289	89,45%
109_March_of_the_Defeated_Underworld_Crypt.nbs	7 159,36	73,26	98,98%	4132	3686	446	89,21%	709	83,87%
10A_Prelude_to_Insanity_Corrupted_Dungeon_Entrance.nbs	427,00	0,00	100,00%	427	427	0	100,00%	0	100,00%
110_Mirage_of_Dust_Sand-Swept_Tomb.nbs	6 371,56	44,56	99,30%	3571	3571	0	100,00%	816	81,40%
111_Dissonance_Ice_Barrow.nbs	1 406,00	16,16	98,85%	1262	1247	15	98,81%	12	99,05%
112_Tribal_Drums_Beating_Undergrowth_Ruins.nbs	4 515,90	20,19	99,55%	2969	2963	6	99,80%	90	97,05%
113_Mind_of_a_Pirate_Galleons_Graveyard.nbs	4 298,93	22,15	99,48%	4020	4009	11	99,73%	417	90,58%
113_Simplified_Mind_of_a_Pirate_Galleons_Graveyard.nbs	3 444,00	5,07	99,85%	2419	2419	0	100,00%	88	96,49%
114_A-U-T-O-M-A-T-O-N_Fallen_Factory.nbs	22 874,40	4 080,35	82,16%	3700	3700	0	100,00%	1660	69,03%
115_Insanity_Corrupted_Dungeon.nbs	3 870,18	33,69	99,13%	4695	4666	29	99,38%	461	91,01%
116_Fierce_Foe_Boss_Battle_1.nbs	4 028,26	26,48	99,34%	2844	2844	0	100,00%	560	83,55%
117_Fearsome_Foe_Boss_Battle_2.nbs	4 834,00	8,30	99,83%	3226	3226	0	100,00%	136	95,95%
118_Chaotic_Encounter_Boss_Battle_3.nbs	4 431,25	19,15	99,57%	3564	3564	0	100,00%	441	88,99%
119_Uncertain_Encounter_Boss_Battle_4.nbs	10 001,84	95,72	99,04%	5044	5030	14	99,72%	1523	76,76%
11A_Out_of_Time_Corrupted_Dungeon.nbs	3 736,67	10,18	99,73%	3833	3770	63	98,36%	24	99,37%
120_Energetic_Encounter_Boss_Battle_5.nbs	8 869,67	53,37	99,40%	5328	5328	0	100,00%	1187	81,78%
120_New_Energetic_Encounter_Boss_Battle_5.nbs	11 710,21	73,25	99,37%	6750	6749	1	99,99%	1612	80,72%
120_New_Part_1_Energetic_Encounter_Boss_Battle_5.nbs	222,82	3,41	98,47%	232	232	0	100,00%	56	80,56%
120_New_Part_2_Energetic_Encounter_Boss_Battle_5.nbs	2 557,57	16,32	99,36%	1755	1755	0	100,00%	365	82,78%
120_New_Part_3_Energetic_Encounter_Boss_Battle_5.nbs	3 169,04	18,74	99,41%	1486	1486	0	100,00%	415	78,17%
120_New_Part_4_Energetic_Encounter_Boss_Battle_5.nbs	34,17	0,02	99,94%	36	35	1	97,22%	0	100,00%
121_Mistress_of_the_Hive_Boss_Battle_6.nbs	9 516,00	72,45	99,24%	5174	5174	0	100,00%	1572	76,70%
122_Melody_of_Reckoning_Boss_Battle_7.nbs	3 029,98	18,07	99,40%	2867	2863	4	99,86%	203	93,38%
123_The_Last_Stand_Boss_Battle_8.nbs	4 805,00	3,89	99,92%	2345	2345	0	100,00%	36	98,49%
124_Weirding_Boss_Battle_9.nbs	5 279,08	60,24	98,86%	4750	4690	60	98,74%	584	88,93%
125_World_Ender_Boss_Battle_10.nbs	4 219,34	34,39	99,18%	4857	4721	136	97,20%	533	89,86%
126_Countdown_to_Corruption_Boss_Battle_11.nbs	68 721,72	3 075,55	95,52%	6906	6872	34	99,51%	12636	35,23%
127_Endless_Climb_Tower_of_Ascension_1.nbs	3 451,64	9,32	99,73%	3406	3404	2	99,94%	53	98,47%
128_Infinite_Ascension_Tower_of_Ascension_2.nbs	15 926,47	130,50	99,18%	7526	7466	60	99,20%	2357	76,01%
129_Unyielding_Influence_The_Nether.nbs	3 932,00	64,70	98,35%	1706	1676	30	98,24%	0	100,00%
130_Begin_the_Siege_Guild_Wars.nbs	3 816,00	4,51	99,88%	2060	2060	0	100,00%	90	95,81%
131_Legendary_Rumble_Legendary_Challenge.nbs	3 569,12	1,14	99,97%	2596	2594	2	99,92%	0	100,00%
132_Bakals_March_Bakals_Theme.nbs	1 558,00	7,20	99,54%	1558	1552	6	99,61%	14	99,11%
133_The_Lass_Hope_886_AP.nbs	5 805,36	16,77	99,71%	3824	3758	66	98,27%	185	95,31%
134_An_Interesting_Find_Discovery_Jingle.nbs	31,00	0,00	100,00%	31	31	0	100,00%	0	100,00%
135_Party_Blues_Party_Bomb.nbs	1 259,00	23,50	98,13%	672	672	0	100,00%	474	58,64%
136_Fringe_of_Fantasy_Pre-Light_Forest.nbs	5 646,68	38,19	99,32%	4054	3951	103	97,46%	533	88,11%
137_Affliction_of_Lunacy_Half-Moon_Island.nbs	3 084,75	8,01	99,74%	2951	2949	2	99,93%	173	94,46%
138_Military_March_Pigmans_Ravines.nbs	3 093,90	101,39	96,72%	2749	2727	22	99,20%	1193	69,57%
139_Stirring_the_Sands_Almuj_Desert.nbs	1 577,45	6,29	99,60%	2119	2054	65	96,93%	0	100,00%
13A_A_Frightening_Find_Silent_Expanse_Discovery_Jingle.nbs	20,89	0,09	99,57%	37	29	8	78,38%	0	100,00%
140_Freedom_in_Darkness_Lutho.nbs	3 640,99	71,44	98,04%	6431	5274	1157	82,01%	372	93,41%
140_Simplified_Freedom_in_Darkness_Lutho.nbs	1 973,35	2,08	99,89%	2251	2251	0	100,00%	15	99,34%
141_All_Eyes_on_Me_Eyeball_Forest.nbs	4 609,93	29,38	99,36%	5979	4961	1018	82,97%	100	98,02%
141_Simplified_All_Eyes_on_Me_Eyeball_Forest.nbs	2 962,82	4,33	99,85%	2312	2312	0	100,00%	47	98,01%
142_Simplified_Where_They_Wandered_Ruined_Olmic_City.nbs	6 153,00	87,63	98,58%	3785	3782	3	99,92%	769	83,10%
142_Where_They_Wandered_Ruined_Olmic_City.nbs	6 854,92	119,85	98,25%	6341	4948	1393	78,03%	840	85,49%
143_Its_Melting_Point_Toxic_Wastes.nbs	3 273,60	93,69	97,14%	5552	4200	1352	75,65%	601	87,48%
143_Simplified_Its_Melting_Point_Toxic_Wastes.nbs	2 688,00	59,91	97,77%	2670	2657	13	99,51%	475	84,83%
144_Simplified_Tangled_in_Endless_Roots_Void_Valley.nbs	2 459,80	62,34	97,47%	2611	2611	0	100,00%	584	81,72%
144_Tangled_in_Endless_Roots_Void_Valley.nbs	2 999,18	90,23	96,99%	5787	4402	1385	76,07%	599	88,02%
145_Everpresence_Eldritch_Outlook.nbs	8 928,32	140,11	98,43%	7455	6114	1321	82,23%	1550	79,78%
145_Simplified_Everpresence_Eldritch_Outlook.nbs	6 737,00	102,48	98,48%	4031	4022	9	99,78%	1260	76,15%
146_End_of_the_Road_The_Portal.nbs	1 941,16	44,35	97,72%	2500	1799	701	71,96%	30	98,36%
147_Genesis_of_the_End_Boss_Battle_12.nbs	33 611,54	262,26	99,22%	14596	14589	7	99,95%	5197	73,73%
147_Loopless_Genesis_of_the_End_Boss_Battle_12.nbs	17 329,47	132,06	99,24%	7548	7544	4	99,95%	2607	74,32%
147_Part_1_Genesis_of_the_End_Boss_Battle_12.nbs	1 060,32	1,51	99,86%	505	504	1	99,80%	27	94,92%
147_Part_2_Genesis_of_the_End_Boss_Battle_12.nbs	5 425,59	57,69	98,94%	2343	2343	0	100,00%	1100	68,05%
147_Part_3_Genesis_of_the_End_Boss_Battle_12.nbs	7 442,20	122,91	98,35%	3287	3287				

Fájlnév	Csend hibája	Hangerőhiba	Hangerő pontos- sága (1 – hiba / csend hibája)	Eredeti felis- hangok száma	Jól mert hangok száma	Nem felis- hangok száma	Jól felismert hangok száma	Csend hang- ként felis- merése	Jól felismert / összes tipp	Jól felismert csend
147_Part_4_Genesis_of_the_End_Boss_Battle_12.nbs	3 401,36	28,88	99,15%	1413	1410	3	99,79%	540	72,31%	42847
148_A_Day_to_Remember_Boss_Battle_13.nbs	26 919,12	289,26	98,93%	12404	12278	126	98,98%	5249	70,05%	404847
148_Old_A_Day_to_Remember_Boss_Battle_13.nbs	10 527,32	133,15	98,74%	5486	5436	50	99,09%	2463	68,82%	171251
148_Part_1_A_Day_to_Remember_Boss_Battle_13.nbs	8 399,32	60,73	99,28%	3715	3664	51	98,63%	1032	78,02%	121353
148_Part_2_A_Day_to_Remember_Boss_Battle_13.nbs	5 051,80	83,10	98,36%	2483	2471	12	99,52%	1578	61,03%	80739
149_Point_of_No_Return.nbs	6 022,92	24,52	99,59%	5734	5563	171	97,02%	132	97,68%	310334
149_Simplified_Point_of_No_Return.nbs	5 349,04	7,74	99,86%	3954	3954	0	100,00%	109	97,32%	310937
150_Skippin_Service_Misadventure_on_the_Sea_1.nbs	2 401,04	74,22	96,91%	2292	2279	13	99,43%	1035	68,77%	105573
151_Lost_at_Sea_Misadventure_on_the_Sea_2.nbs	371,70	6,21	98,33%	889	576	313	64,79%	19	96,81%	31192
151_Simplified_Lost_at_Sea_Misadventure_on_the_Sea_2.nbs	287,00	0,00	100,00%	287	287	0	100,00%	0	100,00%	31813
152_A_Family_Fractured_570_AP.nbs	3 409,13	22,73	99,33%	3803	3623	180	95,27%	282	92,78%	294115
152_Part_1_A_Family_Fractured_570_AP.nbs	445,00	0,83	99,81%	445	445	0	100,00%	12	97,37%	57243
152_Part_2_A_Family_Fractured_570_AP.nbs	1 369,00	22,59	98,35%	1369	1368	1	99,93%	351	79,58%	57580
152_Part_3_A_Family_Fractured_570_AP.nbs	1 080,00	20,60	98,09%	996	993	3	99,70%	356	73,61%	82748
152_Part_4_A_Family_Fractured_570_AP.nbs	515,13	2,79	99,46%	993	816	177	82,18%	0	100,00%	63707
153_Beatskipper_Seaskipper.nbs	1 262,73	7,96	99,37%	374	374	0	100,00%	113	76,80%	26013
154_New_The_Hidden_Horrors_A_Journey_Beyond_1.nbs	2 449,07	4,76	99,81%	1028	990	38	96,30%	4	99,60%	44468
154_Simplified_The_Hidden_Horrors_A_Journey_Beyond_1.nbs	1 980,00	0,87	99,96%	557	557	0	100,00%	0	100,00%	43743
154_The_Hidden_Horrors_A_Journey_Beyond_1.nbs	2 334,18	5,15	99,78%	1028	922	106	89,69%	0	100,00%	44472
155_A_Twisted_Fate_A_Journey_Beyond_2.nbs	496,27	4,32	99,13%	584	535	49	91,61%	21	96,22%	24095
155_Simplified_A_Twisted_Fate_A_Journey_Beyond_2.nbs	423,00	5,73	98,65%	396	395	1	99,75%	20	95,18%	23684
156_A_Spineless_Screech_A_Journey_Beyond_3.nbs	821,68	5,08	99,38%	666	630	36	94,59%	24	96,33%	16610
156_Simplified_A_Spineless_Screech_A_Journey_Beyond_3.nbs	730,00	2,90	99,60%	495	494	1	99,80%	25	95,18%	16480
157_Simplified_What_Must_Be_Done_The_Sacrifice.nbs	223,00	0,00	100,00%	223	223	0	100,00%	0	100,00%	32677
157_What_Must_Be_Done_The_Sacrifice.nbs	271,65	5,65	97,92%	609	440	169	72,25%	0	100,00%	32891
158_Funeral_Service_at_10_OClock_Deaths_Realm.nbs	1 191,00	1,58	99,87%	977	977	0	100,00%	32	96,83%	105891
159_Catalyst_for_a_Fractured_Psyche_Silent_Expanse_Boss_Altar.nbs	11 730,25	173,92	98,52%	12542	12354	188	98,50%	2758	81,75%	336400
160_Earthquake_Boss_Battle_14.nbs	9 832,49	282,22	97,13%	6569	6569	0	100,00%	4542	59,12%	283389
161_Feared_by_Earth_Nest_of_the_Grootslangs.nbs	4 629,88	167,59	96,38%	5048	5046	2	99,96%	3015	62,60%	183637
162_Play_The_Saviour_Boss_Battle_15.nbs	13 568,48	148,60	98,90%	15671	15481	190	98,79%	2461	86,28%	410168
163_Journey_Orphions_Nexus_of_Light.nbs	6 643,16	111,38	98,32%	10280	9208	1072	89,57%	1193	88,53%	256627
163_Simplified_Journey_Orphions_Nexus_of_Light.nbs	5 301,94	80,59	98,48%	6340	6340	0	100,00%	1168	84,44%	260592
164_Face_of_Exinction_Boss_Battle_17.nbs	9 900,98	494,42	95,01%	8320	8320	0	100,00%	8879	48,37%	350201
165_At_Its_Breaking_Point_The_Canyon_Colossus.nbs	6 809,24	57,76	99,15%	7444	7396	48	99,36%	834	89,87%	199422
166_Simplified_The_Light_Fades_Swamp_Discovery.nbs	510,00	0,00	100,00%	492	492	0	100,00%	0	100,00%	63608
166_The_Light_Fades_Swamp_Discovery.nbs	651,42	2,50	99,62%	1005	976	29	97,11%	0	100,00%	63695
167_Burning_Encounter_Boss_Battle_16.nbs	6 952,48	382,26	94,50%	4600	4600	0	100,00%	6068	43,12%	182232
168_For_the_Future_900_AP.nbs	1 753,08	63,58	96,37%	1992	1985	7	99,65%	1152	63,28%	79556
169_Enter_the_Hero_Siegfried_Fanfare.nbs	1 802,04	96,72	94,63%	1881	1881	0	100,00%	1545	54,90%	90274
170_An_Enigmatic_Visit_Gylia_Plains_Discovery.nbs	1 669,84	18,84	98,87%	3600	2564	1036	71,22%	2	99,92%	93098
170_Simplified_An_Enigmatic_Visit_Gylia_Plains_Discovery.nbs	1 397,60	0,27	99,98%	1415	1415	0	100,00%	0	100,00%	94685
171_All_Geared_Up_Avas_Workshop.nbs	1 861,80	47,08	97,47%	1923	1923	0	100,00%	523	78,62%	99854
172_The_Dark_Artists_Anthem_The_Hive.nbs	2 644,16	65,90	97,51%	2822	2789	33	98,83%	385	87,87%	138493
173_Lockdown_Forbidden_Prison.nbs	5 912,72	15,74	99,73%	5453	3937	1516	72,20%	0	100,00%	180247
174_Lights_Hymn_Revamped_Light_Realm.nbs	4 120,82	20,78	99,50%	3664	3644	20	99,45%	314	92,07%	227722
175_Rolling_Overture_Revamped_Cinfras_County.nbs	3 773,72	60,29	98,40%	3742	3723	19	99,49%	816	82,02%	168142
Eight-Legged_Foe.nbs	792,86	0,76	99,90%	2141	2137	4	99,81%	2	99,91%	106757
Összeg	947 530,26	15 130,34	98,40%	631469	611291	20178	96,80%	119846	83,61%	30997785
Minimum			82,16%				64,79%		35,23%	
Minimum fájlnév			114_A...				151_L...		126_C...	
Maximum			100,00%				100,00%		100,00%	
Maximum darabszám						14		96		42