VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

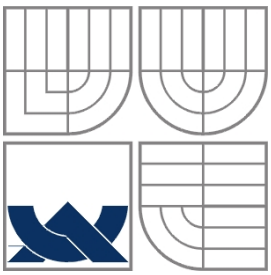
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

TVORBA HUDBY POČÍTAČOM

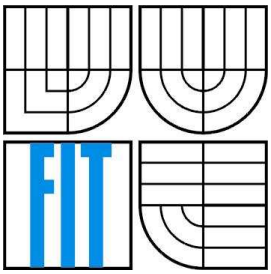
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. PETER MEDERLY
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

TVORBA HUDBY POČÍTAČEM

COMPUTER-GENERATED MUSIC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETER MEDERLY

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL FAPŠO

BRNO 2011

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Mederly Peter, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Tvorba hudby počítačem**
Computer-Generated Music
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte existující přístupy k automatické tvorbě hudby.
2. Vyberte si jeden z nich, proveďte analýzu a návrh systému založeného na daném přístupu.
3. Vytvořte funkční prototyp.
4. Implementujte systém.
5. Prezentujte funkčnost programu na vygenerovaných hudebních ukázkách.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1, 2, 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Fapšo Michal, Ing.**, UPGM FIT VUT
Datum zadání: 20. září 2010
Datum odevzdání: 25. května 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Hlavním cílem práce je analýza, návrh a implementace nového systému schopného generovat hudbu v reálném čase, na základě skutečných dat nebo libovolných rastrových obrázků. Práce poskytuje průřez historií snažení se o formalizaci kompozice hudby a prezentuje současné přístupy k její algoritmické tvorbě. Popisuje techniku programování pomocí omezujících podmínek která je vhodná i pro automatickou kompozici a je součástí implementovaného řešení. Práce do hloubky rozebírá základní komponenty aplikace (rozhraní, generátor, hudební jádro a zvukovou jednotku) a jejich vzájemné interakce. V práci jsou detailně vysvětleny všechny použité postupy a přístupy ke generování a zpracování hudebního materiálu, včetně mnoha průvodních obrázků a ukázkových příkladů. Hudební výstupy jsou podrobeny sérii testů, které odhalují možnosti aplikace a poukazují na její silné i slabé stránky. V závěru je uveden přínos autora v oblasti počítačem generované hudby a vyhlídky pro možná využití a budoucí rozšíření aplikace.

Abstract

The main goal of this thesis is the analysis, design and implementation of a new system, which would be able to generate music in real-time, based on terrain data or any raster image. The thesis deals with history of development of various attempts to formalize musical composition and presents contemporary approaches to its algorithmic creation. Technique of constraint programming is explored as well, because it is suitable also for automatic composition, and it is a part of implemented solution. Application components (interface, generator, music core and sound unit) and their interactions are examined in more detail. All approaches used for generating music are described in depth and, moreover, they are supported by many pictures and practical examples. Musical outputs are tested and results of these tests outline strengths, weaknesses and also inner possibilities of the designed system. Conclusion summarises author’s contribution to the field of computer generated music and reveals possible prospects for application usage and its extensions.

Klíčová slova

hudba, generování, kompozice, algoritmická, problémy s omezujícimi podmínkami, CSP, agent, terén, MIDI, OSC, Strasheela

Keywords

music, generation, composition, algorithmic, constriant satisfacion problem, CSP, agent, terrain, MIDI, OSC, Strasheela

Citace

Mederly Peter: Tvorba hudby počítačom, diplomová práce, Brno, FIT VUT v Brně, 2011

Tvorba hudby počítačom

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michala Fapša. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Peter Mederly
25.5.2011

Obsah

1 Úvod

Je každodennou súčasťou nášho života a bola ňou už od nepamäti. Hudba sú zvuky s rozdieľnymi výškami usporiadané do melódií a organizované do rytmických vzorov a metra [1]. Nikto presne nevie, kedy a ako vznikla, kto ju „vymyslel“. Vieme však, že už tisícročia je pre ľudí zaujímavá a hľadáme v nej potešenie aj útechu.

Aby sme v súčasnosti mohli hudbu generovať automaticky, musíme sa najprv vrátiť až do obdobia antiky. Grécky filozof, matematik a hudobný teoretik Pythagoras zdokumentoval vzťah medzi matematikou a hudbou a položil tak základy dnešnej hudobnej teórie a akustiky [2]. Jednou z jeho základných koncepcií, z ktorej vychádza dnešná hudobná teória, je tzv. „hudba sfér“. Podľa Romana Dykasta sú v kozmickej harmónii výšky tónov určené rýchlosťou pohybu jednotlivých planét, pričom rýchlosť je určená ich vzájomnými vzdialenosťami. Týmto kosmickým pomerom mali zodpovedať aj pomery, z ktorých v hudbe vznikajú harmonické intervaly v rámci oktávy. Proporcie medzi tónmi majú matematickú povahu a symfonické môžu byť iba tie intervaly, ktoré je možné vyjadriť najjednoduchším matematickým pomerom malých kladných čísel (n + 1 / n). Tomuto pomeru zodpovedajú tri intervaly: oktáva (2:1), kvinta (3:2) a kvarta (4:3). Kozmickú hudbu, harmóniu sfér, však ľudské ucho zachytiť nemôže. Pytagorovci to prirovnávali k situácii človeka, ktorý od narodenia žije pri morskom brehu a prestáva vnímať nepretržitý hluk vĺn, lámajúcich sa o breh. Hudbu sfér nepočujeme, pretože znie takisto nepretržite [3, str.84]. Myšlienky Pytagora a jeho nasledovníkov inšpirujú dodnes, aj keď už boli prekonané.

Významným krokom vpred vo vývoji hudby bolo vynájdenie modernej hudobnej notácie, ktorá používa notovú osnovu. Tento objav je uznaný talienskemu benediktínskemu mníchovy Guido D’Arezzovi a bol učinený okolo roku 1026. Umožňil skladateľom používať notový zápis a ich skladby tak mohol rovnako interpretovať aj iný hudobník.

V renesancii, v období holandskej polyfónie, bola obľúbená kontrapunktická forma kánon. Je založená na princípe prísnej imitácie hlavnej melódie [4, str.81]. To znamená, že boli ustanovené striktné pravidlá pre jeho tvorbu. Druhý hlas musí byť presná kópia alebo kontrapunktická derivácia prvého hlasu, druhý hlas musí začať neskôr ako prvý a nemôže sa od neho a jeho kontrapunktických variácií odchyľovať [5]. Vrcholnou formou kontrapunktu je fuga. Jeden z najznámejších hudobných skladateľov J. S. Bach dokonca „vstaval“ svoje meno do skladby „Umenie fugy, Fuga XV“.

V druhej polovici 18. storočia sa v Európe objavili hudobné hry s kockami. Najznámejším príkladom je „Musikalisches Würfelspiel“ od neznámeho autora, viacerí vydavatelia však za autora označili Mozarta [6, str.27]. Princíp hier bol veľmi podobný, namiesto jednej noty sa náhodne (hodom kocky alebo iným spôsobom) vybral vopred napísaný úsek, napr. jeden takt. Úseky boli spájané za seba lineárne, aby vytvorili skladbu.

S nástupom počítačovej éry sa začala oblasť algoritmickej kompozície skúmať ešte intenzívnejšie. Vzniklo mnoho rôznych prístupov a aplikácií, ktoré slúžia buď ako podporné (inšpiratívne) prvky pre hudobného skladateľa alebo sa ho snažia nahradiť sami. Súčasné prístupy sú rozpísané v kapitole 2.1.

Za najstaršie známe nahrávky počítačom generovanej hudby sa považujú piesne „Baa Baa Black Sheep“ a orezaná verzia „In the Mood“ [7]. Nahrávky vznikly v roku 1951 na University of Manchaster, ich zdrojom bol počítač Ferrari Mark I, ktorý bol komerčnou verziou tzv. „Baby Machine“.

1.1 Cieľ a poňatie práce

Táto práca má za úlohu priblížiť čitateľovi algoritmickú kompozíciu hudby, tj. tvorbu hudby počítačom. Úvodom som problematiku načrtol z filozofického a historického hľadiska, mojou ďalšou úlohou je v rozumnej miere prezentovať potrebné teoretické základy. Práca nepokryje celú tematiku hudobnej kompozície ani algoritmického generovania hudby, ale poskytne dostatok referencií a odkazov na vhodné zdroje k ďalšiemu štúdiu.

Cieľom práce je tiež zoznámenie sa s existujúcimi prístupmi k algoritmickej kompozícii a návrh a implementácia nového systému pre automatickú tvorbu hudby. Vstupné dáta budú predané hudobnému jadru, kde budú spracované a prehrané vo zvukovej jednotke v reálnom čase. Systém má byť „priateľský k užívateľom“ a súčasne má poskytovať mnoho možností pre generovanie hudby.

1.2 Prehľad obsahu kapitol

Práca je rozdelená na päť hlavných kapitol. Prvá kapitola vymedzuje problematiku z pohľadu teórie. Čitateľ získa prehľad v existujúcich prístupoch využívaných pri algoritmickej kompozícii hudby. Tie spadajú do kategórie prístupov stochastických alebo založených na pravidlách, či umelej inteligencii. Ďalej popisuje možnosti MIDI protokolu a techniky zvanej „constrained programming“.

Kapitola druhá obsahuje analýzu požiadaviek na navrhovaný systém. Využíva analýzu zhora dole a rozdeľuje ho na tri celky. Každý z nich je nakoniec rozobraný podrobnejšie. Do tejto kapitoly spadá aj výber vývojového prosredia a implementačných nástrojov.

Tretia kapitola pojednáva o návrhu aplikácie. Každá zanalyzovaná časť systému je rozobraná do väčšej hĺbky a sú navrhnuté postupy a metódy, ktoré budú použité pri implementácii. Venuje sa napr. získavaniu a interpretácii vstupných dát a komunikácii medzi prvkami aplikácie.

Samotná implementácia systému pre tvorbu hudby počítačom je rozobraná v štvrtej kapitole. V texte je rozpísané použité rozhranie a k nemu pripojené komponenty generátoru, hudobného jadra a zvukovej jednotky. Každý komponent má svoju vlastnú podkapitolu, v ktorej sú podrobne

vysvetlené všetky aspekty jej implementácie. Text je doplnený množstvom obrázkov, ktoré čitateľovi uľahčia pochopenie tematiky a použitých princípov.

Záverečná kapitola obsahuje zhodnotenie dosiahnutých výsledkov pomocou série testov a porovnaní vygenerovaného hudobného materiálu. Zároveň poukazuje na silné a slabé stránky implementovaného riešenia a vysvetľuje ich možné príčiny.

2 Teoretické vymedzenie problematiky

V tomto texte nie sú popísané žiadne poznatky z teórie hudby alebo princípov kompozície. Dôvodom je, že pri implementácii systému si vystačíme so základnými znalosťami hudobnej teórie, ako sú vlastnosti tónu, rôzne druhy stupníc a melodické či harmonické intervaly. Tie u čitateľa, ktorý siahol po tejto práci, predpokladám.

Kapitola rozoberá dôležité teoretické znalosti z oblasti automatickej tvorby hudby. Predstavím existujúce prístupy k algoritmickej kompozícii a základy MIDI protokolu. V poslednej časti kapitoly je rozpísaná technika „constraint programming“, vhodná ku generovaniu hudby pomocou sady obmedzujúcich pravidiel.

2.1 Automatická tvorba hudby

Jednotlivé prístupy k automatickej tvorbe hudby sa dajú rozdeliť do troch kategórií. Stochastické prístupy sú založené na náhode, ďalšie sú založené na pravidlách alebo aplikujú poznatky z oblasti umelej inteligencie. Existujú aj hybridné metódy, ktoré kombinujú prvky z rôznych oblastí. Ku každému prístupu je uvedený krátky neformálny teoretický popis a príklad.

Okrem prístupu k tvorbe hudby je zaujímavé aj to, z čoho sa dá vytvoriť. Pri štúdiu problematiky som narazil na prácu Johna Dunna, ktorý pomocou softwareu Algoritmic Arts zhudobnil DNA [8]. Ďalším zaujímavým príkladom je použitie dát z burzy v práci Samuela Vana Ransbeck a Carlosa Guedesa [9].

2.1.1 Stochastické prístupy

Základný stavebný prvok týchto systémov je náhoda a výsledok preto nemôže byť vopred známy. Najjednoduchší prípad stochastickej tvorby hudby je generovanie náhodných sekvencií nôt. Tento prístup sa podobá tvorbe dadaistickej literatúry, keď sa slová len tak ťahali z klobúka. Existuje viacero možností ako využiť náhodu pri generovaní hudby sofistikovanejším spôsobom. Každý z nižšie uvedených prístupov ju využíva trochu inak.

Markovove reťazce

V roku 1907 začal A. Markov študovať nový dôležitý typ náhodného procesu. Skúmal, ako výsledok predošlého experimentu môže ovplyvniť výsledok nasledujúceho experimentu. Markovove reťazce sú diskrétne pravdepodobnostné systémy, kde pravdepodobnosť budúcej udalosti závisí na jednej alebo viacerých minulých udalostiach [10].

Markovov reťazec definujeme ako konečnú množinu stavov S a prechodovú maticu P. Proces začína v jednom zo stavov z množiny S. Ak sa nachádza v stave s_i, tak do stavu s_j sa dostane s pravdepodobnosťou p_{ij}. Pre počet minulých udalostí, ktoré sa berú do úvahy sa zaužíval termín rád Markovovho reťazca. Tabuľka 2.1 predstavuje prechodovú maticu reťazca prvého rádu [11]. Pre hlbšie pochopenie problematiky Markovovych reťazcov alebo náhody ako takej odporúčam [12, str. 405].

	C4	D4	E4	F4	G4	A4	B4	C5
C4	.375	.25		.125	.125			.125
D4	.66						.33	
E4	.5	.5						
F4	.25		.5		.25			
G4				1				
A4				1				
B4						.5	.5	
C5						1		

Tabuľka 2.1 – Prechodová matica Markovovho reťazca prvého rádu

V tabuľke 2.1 máme uvedený príklad prechodovej matice, ktorý definuje pravdepodobnosti prechodov medzi tónmi v štvrtej oktáve. Do úvahy sa berie iba jeden predošlý tón (reťazec prvého rádu). Napr. z tónu C4 sa s pravdepodobnosťou 25% dostaneme na tón D4, s pravdepodobnosťou 12,5% na tóny F4, G4, či C5 a s pravdepodobnosťou 37,5% v ňom zostaneme. Pozornému čitateľovi určite neuniklo, že súčet pravdepodobností v jednom riadku tabuľky je vždy 100%. Vo väčšine prípadov sa pravdepodobnosti prechodov tvoria analýzou existujúcich skladieb. Systém potom produkuje výstup, ktorý je štýlom podobný vstupu.

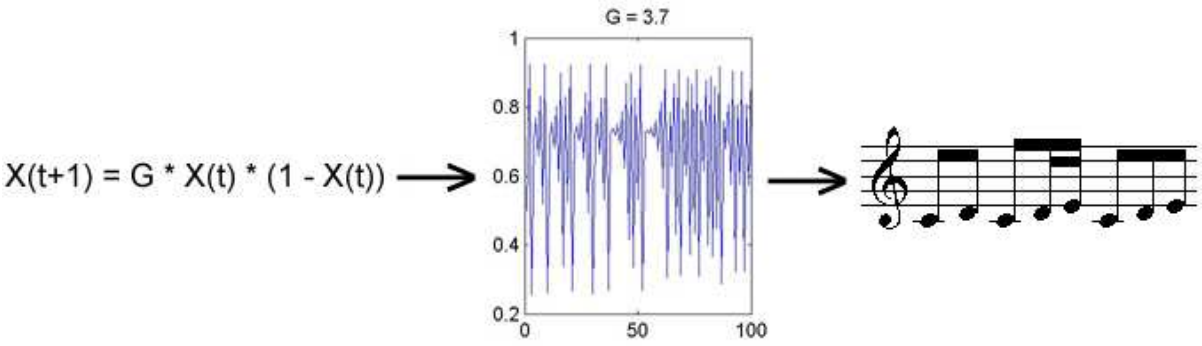
Chaos a sebepodobné systémy

Termín chaos popisuje nezrovnalosti a nelineárnosti mnohých prírodných fenoménov porovnávaných s teóriami a rovnicami, ktoré ich popisujú [13]. Chaos sa modeluje pomocou nelineárnych dynamických systémov (matematických rovníc). Tieto systémy môžu vykazovať usporiadané alebo chaotické správanie, ktoré sa v čase sa nedá predpovedať. Ich spoločným znakom je vysoká citlivosť na zmeny hodnôt vstupných parametrov.

Príťažlivé sú vďaka svojim fraktálnym vlastnostiam. Sebepodobnosť vzniká pripojením výstupu systému späť na vstup, napr. takto:

$$y(t+1) = f(y(t))$$

V chaotickej oblasti nelineárneho systému sa vytvárajú takmer zhodné a skoro periodické vzory. Pri automatickom generovaní hudby sa používajú práve tieto kvázi periodické vzory, ktoré sa začnú objavovať pri určitej hodnote vstupných parametrov [14]. Kombináciou náhodnosti a opakovateľnosti vznikajú zaujímavé hudobné motívy. Na obrázku 2.1 je znázornená logistická rovnica, odozva systému v čase pre parameter G rovný hodnote 3,7 a potenciálny hudobný výstup.



Obrázok 2.1 – Odozva logistickej rovnice a jej možná hudobná interpretácia

2.1.2 Prístupy založené na pravidlách

Pre ľudí je prirodzená snaha analyzovať a definovať veci vo svojom okolí. Predsa aj jazyk, ktorým hovoríme má gramatiku, ktorú tvorí sada pravidiel. Aplikácia pravidiel pri automatickej kompozícii je veľmi rozšírená a aj ja som pre navrhovanú aplikáciu zvolil takýto prístup.

Formálne gramatiky a automaty

Gramatiku definujeme ako štvoricu (N, Σ, P, S), kde N je množina neterminálnych symbolov, Σ je množina terminálnych symbolov, P je množina prepisovacích pravidiel a S je začiatočný symbol. Množiny N a Σ sú disjunktné a symbol S patrí do množiny N. Gramatika predstavuje generatívny systém, v ktorom aplikáciou prepisovacích pravidiel na neterminálne symboly dostaneme reťazec zložený len z terminálnych symbolov. Práve takéto reťazce reprezentujú vety gramatikou definovaného jazyka. Existujú štyri typy gramatík, ktoré sú rozlíšené na základe ich rozhodovacej sily. Ku každému typu gramatiky môže byť zostrojený odpovedajúci automat, v tomto texte však nebudem úvadzať definície automatov a ani dôkazy prevodov medzi gramatikou a automatom.

Gramatiky sú vhodným prostriedkom ako zachytiť hierarchickú štruktúru hudby. Skladateľ nadefinuje hudobnú gramatiku a bude generovať skladby (vety jazyka). Neterminálne symboly generujú štruktúru skladby, kým terminálne symboly predstavujú najnižšie jednotky v hierarchii, napr. noty. Formát prepisovacích pravidiel závisí na type zvolenej gramatiky.

Constraint Programming

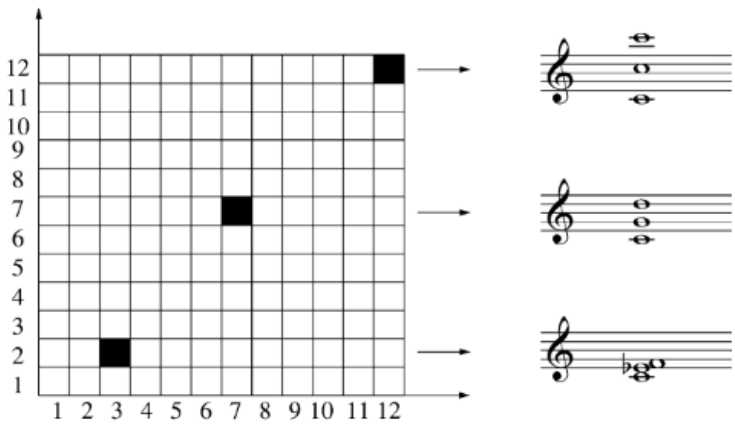
Tejto metóde programovania a jej aplikácii pri algoritmickom generovaní hudby je venovaná celá kapitola 2.3, pretože som sa ju rozhodol použiť v navrhovanom systéme.

Celulárne automaty

Celulárne automaty sú dynamické systémy diskrétne v čase a priestore, ktoré pracujú v pravidelnej mriežke. Sú pre nich charakteristické lokálne interakcie, takže každá bunka pozná iba svoje okolie. V každom časovom kroku sa súčasne aktualizujú všetky bunky. Nový stav sa nastaví na základe aktuálneho stavu bunky a jej okolia. Tento prechod je definovaný pravidlom alebo funkciou.

Celulárny automat pre generovanie hudobnej štruktúry použil ako prvý Peter Beyls. Jeho riešenie zahŕňalo pridanie ďalších metód pre definovanie okolia a zapojenie viacerých minulých stavov bunky do výpočtu nového stavu. Bol jednorozmerný a tóny boli do buniek priradené uživateľom. Ak sa vygeneroval rovnaký tón ako bol ten predošlý, nedošlo k jeho uvoľneniu, čo dodávalo výslednej skladbe určitý rytmus [15].

Ďalším príkladom je software CAMUS. Jeho autor Eduardo Miranda použil dva celulárne automaty. Jeden generoval výšku a dĺžku tónu a druhý poskytoval parametre pre inštrumentalizáciu štruktúry. Systém napr. interpretoval súradnice buniek v mriežke ako trojice nôt. Na súradnici x je interval medzi spodným a stredným tónom akordu, na súradnici y interval medzi stredným a horným tónom (obrázok 2.2).



Obrázok 2.2 – Mapovanie akordov v CAMUS

2.1.3 Umelá inteligencia

Do tejto kategórie spadajú systémy, ktoré sa dokážu „učiť“. Podobne ako prístupy z kapitoly 2.1.2 majú definované pravidlá alebo gramatiku, ale sú schopné definovať a osvojiť si vlastné pravidlá.

Prechodové siete

Prechodové siete (transition networks) sú zostrojené ako množina konečných automatov a sú reprezentované grafom. Jeho hrany označujú prechody a uzly stavy jednotlivých automatov. Každý automat predstavuje neterminálny symbol a je tvorený vlastnou sieťou. Hrany týchto sietí sú označené terminálnym symbolom (koncové stavy) alebo neterminálom, ktorý odkazuje na inú sieť. Zo začiatočnej siete a jej začiatočného uzlu postupne prechádzame po jednotlivých hranách. Ak sa narazí na neterminálny symbol, systém sa ponorí do odpovedajúcej siete a takto pokračuje až kým sa nenahradia všetky neterminály. Špeciálnym typom prechodových sieti sú Petriho siete, ktoré simulujú udalosťami kontrolované procesy [15].

Systém EMI (Experiments in Musical Intelligence) je veľmi známym príkladom algoritmickej kompozície. V roku 1981 ho navrhol David Cope a je schopný sa „oboznámiť“ so štruktúrou skladieb a tvoriť podobné výstupy. EMI využíva ATN (augmented transition network) a rôzne ďalšie stratégie k analýze mnohých aspektov hudobného materiálu. EMI využíva tri základné princípy. Najprv rozoberá a analyzuje skladby a ich prvky. Z nich extrahuje syntaktické a sémantické pravidlá, ktoré zachovávajú štýl analyzovaných skladieb. Na záver ich rekombinuje a skladá dohromady.

Umelé neurónové siete

Podobne ako evolučné algoritmy, spomínané v ďalšej kapitole, nachádzajú umelé neurónové siete inšpiráciu v biológii. Sieť tvoria vzájomne poprepájané prvky zvané neuróny. Tie sú usporiadané aspoň do dvoch vrstiev, vstupnej a výstupnej. Väčšinou sa medzi nimi nachádza niekoľko ďalších skrytých vrstiev. Spôsob, akým sú vrstvy medzi sebou poprepájané určuje topológia siete. Propagačná funkcia kombinuje všetky vstupy daného neurónu do jedného a aktivačná funkcia potom na základe prahu určí jeho stav. Nový stav sa privedie na vstup pripojených neurónov. Vstupy do neurónu maju svoje váhy, práve ich modifikáciou sa sieť „učí“. Trénovacie dáta slúžia k vhodnému nastaveniu váh. Sieť sa snaží zapamätať si charakteristiku dát predložením vstupu aj správneho výstupu. Ak sa výstup siete nepodobá na poskytnutý výstup, sú modifikované jej váhy a proces sa opakuje. Po naučení by sieť mala poznať charakteristiky vstupných dát z trénovacej množiny a správne ich aplikovať.

Neurónové siete sa využívajú v množstve rôznych aplikácií, napr. pre rozpoznávanie vzorov, automatickú klasifikáciu a optimalizáciu. V oblasti automatickej tvorby hudby vznikol napr. systém CONCERT. Jeho autorom je Michael Mozer a produkuje melódie so základnou harmonickou postupnosťou [15]. Systém je trénovaný na Bachovych chorálоch, folkových melódiách a harmonických postupnostiach valčíkov. Na tomto systéme je zaujímavé napr. to, že výstup nie je interpretovaný ako jedna nota, ale môže pozostávať z pravdepodobností pre výber nasledujúcej noty.

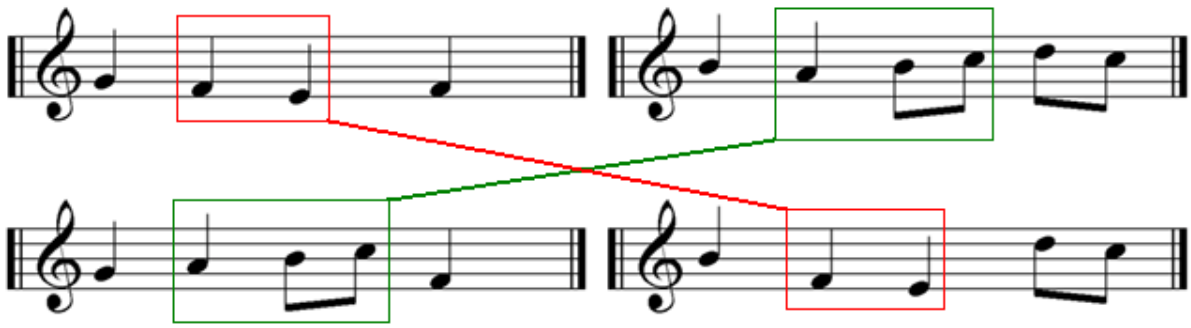
2.1.4 Hybridné prístupy

Do tejto kategórie spadajú všetky prístupy, ktoré ľubovoľným spôsobom kombinujú stochastické metódy, pravidlá a prvky umelej inteligencie. Keďže si môžu brať kúsok z každej oblasti, je tu priestor pre zaujímavé riešenia.

Evolučné algoritmy

Sú to stochastické prehľadávacie metódy založené na Darwinovej teórii o evolúcii. Podobne ako v biológii máme populáciu jedincov (potenciálnych riešení), pričom do ďalšej generácie sa dostanú najlepší potomkovia. Začiatočná populácia môže byť napr. vygenerovaná náhodne alebo zadaná užívateľom. V každej generácii dochádza k „biologickým“ operáciám ako je kríženie a prípadne k mutáciám. Možností pre výber jedincov je viacero, mali by však vyberať jedinca na základe tzv. „fitness“ funkcie. Tá je najdôležitejším prvkom evolučných algoritmov a určuje, ako je daný jedinec vhodný pre ďalší výpočet.

Pre lepšiu predstavu uvediem príklad. Pri generovaní hudby by mohla byť začiatočná populácia vygenerovaná ako náhodná postupnosť nôt rôznych výšok a dĺžok. Kríženie by predstavovalo výmenu sekvencie nôt dvoch jedincov, mutácia by menila výšku a prípadne dĺžku nôt. Najjednoduchším spôsobom ako ohodnotiť kvalitu jedincov je ich vypočutie koncovým užívateľom. Ten na základe svojho subjektívneho pocitu potomka ohodnotí. Na obrázku 2.3 je znázornené kríženie hudobného materiálu spôsobom, aký bol uvedený v tomto príklade. Ohodnotiť by sa dali napríklad aj naučenou neurónovou sieťou, ktorá by hodnotila, ako dobre potomok generuje požadovaný hudobný štýl.



Obrázok 2.3 – Kríženie hudobného materiálu

2.2 MIDI

MIDI je skratka odvodená z názvu Musical Instrument Digital Interface, voľne preložiteľného ako digitálne rozhranie pre hudobné nástroje. Názov MIDI môžeme používať v troch rozdielnych významoch: fyzický konektor, prenosový protokol a formát súboru. Pre pochopenie tejto práce je

dôležité poznať MIDI protokol a práve jemu je táto kapitola venovaná. Nasledujúce informácie sú prevzaté z [16] a [17].

MIDI protokol pozostáva zo správ, ktoré majú dĺžku jeden, dva, alebo tri bajty. V jednom prípade môže mať správa dokonca neobmedzený počet bajtov. Prvý bajt je vždy stavový a ako jediný má nastavený najvyšší bit na hodnotu 1. Tým je umožnené ľahko detekovať začiatok každej správy. Bajty prijaté za stavovým bajtom nazývame dátové. Existujú správy hlasového kanála (channel voice), bežné systémové (system common) a systémové v reálnom čase (system realtime). Systémové správy v reálnom čase sú určené pre všetky zapojené nástroje a zariadenia, oproti bežným systémovým správam ale obsahujú iba status bajt a sú určené pre synchronizáciu zariadení. Prvý typ správy je najbežnejší a zároveň najzaujímavejší. Záhŕňa napr. správy typu aktivácie a deaktivácie tónu. Všetky správy hlasového kanálu sa dajú odvodiť z tabuľky 2.2.

Status Byte	Data Byte 1	Data Byte 2	Message	Legend
1000nnnn	0kkkkkkk	0vvvvvvv	Note Off	n = channel (0-15) k = key (0-127) v = velocity (0-127) p = pressure (0-127) c = controller u = controller value(0-127) e = preset number (0-127) r = coarse f = fine
1001nnnn	0kkkkkkk	0vvvvvvv	Note On	
1010nnnn	0kkkkkkk	0ppppppp	Poly Key Pressure	
1011nnnn	0ccccccc	0uuuuuuu	Controller Change	
1100nnnn	0eeeeeee	[none]	Program Change	
1101nnnn	0ppppppp	[none]	Channel Pressure	
1110nnnn	0fffffff	0rrrrrrr	Pitch Bend	

Tabuľka 2.2 – Formát MIDI správ hlasového kanála

V ďalšom texte budú často spomínané správy o stlačení alebo uvoľnení tónu a budú používané aj ich anglické názvy NOTE ON a NOTE OFF. Hodnoty prenášané v dátových bajtoch majú rozsah 0-127, pretože ich najvyšší bit musí mať hodnotu 0. Je vhodné spomenúť fakt, že čísla MIDI kanálov sú interne číslované od nuly, ale všetky zariadenia zobrazujú čísla kanálov od jednotky. Existuje až 16 rôznych MIDI kanálov, ktoré sa dajú využiť pre posielanie správ rôznych hudobných nástrojov.

2.3 Constraint programming

Constraint programming (voľne preložiteľné ako programovanie pomocou obmedzení) je prístup, ktorý využíva techniky pre riešenie tzv. „constraint satisfaction problems“. Tieto problémy splnenia obmedzujúcich podmienok budú v ďalšom texte označované skratkou CSP (odvodenou z anglického názvu). Problematika je popísaná neformálne, v závere kapitoly sú uvedené zdroje, z ktorých som čerpal.

CSP pozostáva z množiny premenných a matematických vzťahov medzi nimi, označovaných ako obmedzenia. CSP v podstate predstavuje kombinačný problém a riešenie nemusí existovať,

prípadne ich môže byť viac ako len jedno. Riešením sa rozumie výber hodnoty všetkým premenným z ich oborov hodnôt, ktoré zároveň spĺňajú všetky definované obmedzenia.

Pre automatické riešenie týchto problémov je potrebný špeciálny software. S bežnými programovacími prístupmi sa CSP nedá uspokojivo vyjadriť a ešte ťažšie sa hľadá jeho riešenie. V programovacích jazykoch ako napr. C a Java má premenná priradenú nejakú konkrétnu hodnotu. V kontexte programovania pomocou obmedzení je premenná chápaná skôr ako logicky kvantifikovateľná premenná v predikátovej logike prvého rádu [18] a nemení svoju hodnotu. Takáto premenná má svoj obor hodnôt, ktoré môže v riešení nadobúdať. Vo väčšine prípadov ide o konečnú množinu prvkov rovnakého typu.

Pre nájdenie riešenia CSP je potrebné prehľadávať stavový priestor všetkých možných riešení, ktorý býva často veľmi rozsiahly. Preto je potrebný rýchly a efektívny „riešiteľ obmedzení“ (constraint solver). Pre riešenie CSP sa často používa spätné trasovanie (backtracking). Keď premenná nemá na priradenie žiadnu prípustnú hodnotu zo svojho oboru, výpočet sa vráti o krok späť a skúsi inú cestu. Medzi bežné spôsoby ako urýchliť výpočet patrí zoraďovanie premenných a hodnôt a propagovanie informácií cez obmedzenia. Ich prínos je včasné identifikovanie neúspešných prípadov prehľadávania.

Jednou z metód zoraďovania premenných je prístup založený na výbere premenných s najmenšiou množinou prípustných hodnôt (označovaná aj ako „fail-first“ heuristika). To znamená, že ako prvá sa vyberá premenná, ktorá s najväčšou pravdepodobnosťou spôsobí neúspech prehľadávania. Ďalším prístupom je heuristika miery (degree heuristic). Ako prvá sa vyberie premenná, ktorá je obsiahnutá v najväčšom počte obmedzení pre ostatné nepriradené premenné. Dôležité je aj poradie vyberania hodnoty z oboru hodnôt zvolenej premennej. Výberom takej, ktorá najmenej obmedzí hodnoty ostatných premenných sa zachová maximálna flexibilita. Posledný spomínaný prístup neovplyvní výsledok ani rýchlosť hľadania ak sa snažíme nájsť všetky možné riešenia alebo ak neexistuje žiadne riešenie (musia sa vyskúšať všetky možnosti).

Veľkosť prehľadávaného priestoru sa dá zmenšiť propagáciou obmedzujúcich podmienok, napr. kontrolou smerom dopredu (forward checking). Keď nejakej premennej určíme hodnotu, forward checking obmedzí všetky ostatné s ňou spojené premenné – odstránením hodnôt z ich oboru, ktoré sa vylučujú so zvolenou hodnotou premennej. Ďalším spôsobom je test kruhovej konzistencie (arc consistency), ktorý dokáže odhaliť viac prípadov ako spomínaný forward checking.

Okrem spätného trasovania sa používa aj lokálne prehľadávanie spolu s heuristikou minimálneho počtu konfliktov. Tá je v niektorých prípadoch CSP veľmi účinná, hlavne pri plánovacích úlohách. Výhoda lokálneho prehľadávania je zjavná pri malej zmene zadania problému. Z predchádzajúceho riešenia sa dokáže v malom počte krokov dopracovať k novému riešeniu a to s minimálnym počtom zmien. Naopak pri spätnom trasovaní sa musí celý proces zopakovať a môže dospieť k úplne inému riešeniu.

Podrobnejšie a formálnejšie vysvetlenie CSP môže čitateľ nájsť v knihe od Barkleyho [19]. Použitie techniky programovania pomocou obmedzení v oblasti algoritmickej tvorby hudby je výborným spôsobom vysvetlené v [20], kapitola 3.

3 Analýza problému

Po priblížení teoretického zázemia problematiky generovania hudby sa môžeme pustiť do analýzy skúmaného problému. Pri tvorbe ľubovoľného projektu sa oplatí nachvíľu pozastaviť a položiť si zopár základných otázok. Čo sa od systému očakáva? Ako má byť stavaný? Čo sú jeho najdôležitejšie časti? A v neposlednom rade, pre koho je určený?

Táto kapitola slúži k nájdeniu všetkých potrebných odpovedí ešte skôr, ako sa pristúpi k samotnému návrhu systému. Prvou úlohou je ujasniť si všetky kritéria, ktoré by mali byť splnené vo finálnom prevedení. Tie záležia hlavne na cieľovej skupine užívateľov. Ďalší významný krok je rozdelenie problému do viacerých logických celkov. Bude tak jednoduchší na pochopenie a zároveň sa bude ľahšie implementovať. Nakoniec, po vytvorení ucelenej predstavy o riešenej problematike, je možné zvoliť vhodné implementačné prostredie a vývojové nástroje.

3.1 Požiadavky na systém

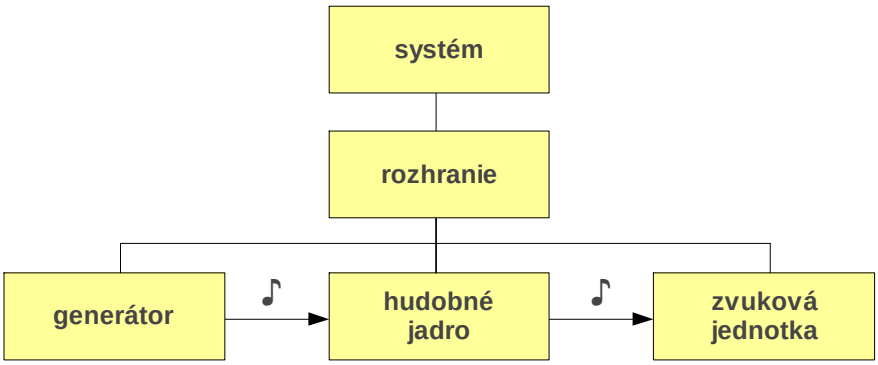
Táto práca sa zaoberá špecifickým spektrom informačných technológií. Mnou navrhovaný systém pokrýva už len vymedzenú časť z tejto oblasti. Napriek tomu by vytvorená aplikácia mala byť prístupná aj širšej, laickej verejnosti a nie len odborníkom. Hlavné požiadavky sú:

- Desktopová aplikácia umožňujúca generovanie hudby
  - komponenty komunikujú pomocou správ
  - možnosť vlastnej implementácie komponentov
  - prehľadné užívateľské rozhranie
  - vytváranie, ukladanie a načítavanie vlastných nastavení
  - nápoveda s tutoriálom
  - dokumentácia
- Vstupné dáta
  - podpora rôznych formátov
  - načítavanie viacerých súborov
    - dáta organizované do vrstiev
    - každá vrstva má definované metadáta
- Zvukový výstup
  - vo formáte MIDI
  - prehrávaný v reálnom čase
  - možnosť zapísať do súboru

3.2 Analýza zhora dole

Ako napovedá názov kapitoly, riešený problém bude rozdelený do menších a menších celkov, až na základné elementy alebo operácie. Každá úroveň podproblémov vyjadruje iný stupeň abstrakcie, najväčšia miera abstrakcie je na najvyššej úrovni. Tento jednoduchý prístup je vo svete široko využívaný, a to nie len v oblasti informačných technológií.

Na najvyššej úrovni sa systém pre generovanie hudby skladá z rozhrania, na ktoré sú pripojené tri základné celky: generátor, hudobné jadro a zvuková jednotka (obrázok 3.1). Generátor produkuje noty v „surovej“ podobe pre hudobné jadro, ktoré ich spracuváva. Z jadra sa noty posielajú do zvukovej jednotky. Tá ich prehrá alebo uloží do súboru. Jednotlivé celky sú ďalej rozobrané v nasledujúcich podkapitolách.

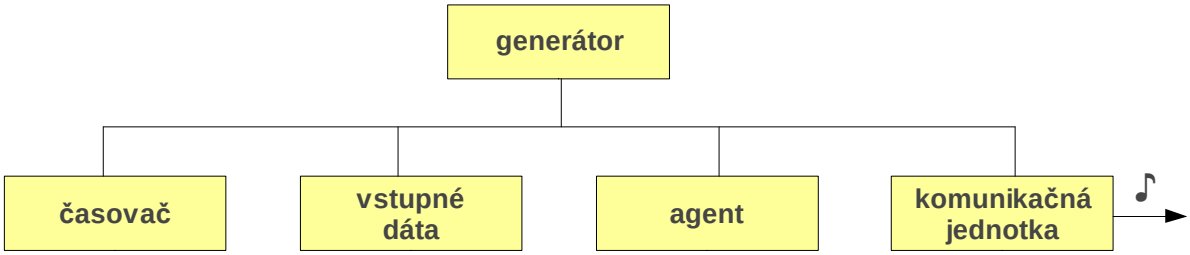


Obrázok 3.1 – Najvyššia abstraktná úroveň systému

3.2.1 Generátor

V kapitole 2.1 bolo naznačených mnoho prístupov ako automaticky generovať hudbu. Každý takýto systém musí nejakým spôsobom poskytnúť dáta, ktoré budú vhodne interpretované vo výslednom hudobnom diele. Ani tento prípad nie je výnimkou.

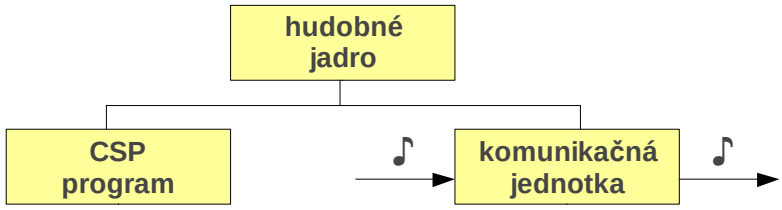
Generátor bude mať k dispozícii časovač, ktorý v určených intervaloch zabezpečí odoslanie správ pre hudobné jadro. Správy bude vytvárať agent, pracujúci so vstupnými dátami. Zároveň ich bude poskytovať komunikačnej jednotke na odoslanie. Jednotlivé elementy generátora sú znázornené na obrázku 3.2.



Obrázok 3.2 – Analýza generátora

3.2.2 Hudobné jadro

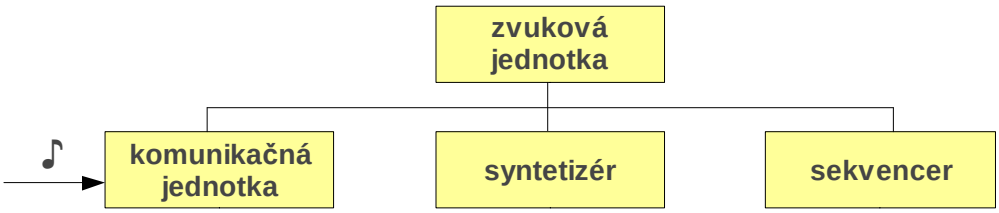
Úlohou hudobného jadra je previesť z veľkej miery náhodný vstup z generátora do počúvateľnej podoby. Na obrázku 3.3 je znázornená jeho štruktúra. Komunikačná jednotka prijme správu z generátora a predá ju CSP programu (kapitola 2.3). Po vyriešení problému sa výsledok pošle komunikačnou jednotkou do zvukovej jednotky.



Obrázok 3.3 – Analýza hudobného jadra

3.2.3 Zvuková jednotka

Komunikačná jednotka zvukového modulu čaká na správy z hudobného jadra. Po ich prijatí sú okamžite prehrané syntetizérom (prehrávanie v reálnom čase). Sekvencer správy tiež zaznamená pre prípad, že bude potrebné zapísať zvukový výstup do súboru.



Obrázok 3.4 – Analýza zvukovej jednotky

3.3 Výber prostredia a nástrojov

Výber implementačných nástrojov, vývojového prostredia a operačného systému je ďalší významný krok analýzy systému. Ak má byť aplikácia dostupná čo najväčšej skupine koncových užívateľov, musí byť prenositeľná a podľa možností by mala využívať iba programy s otvoreným zdrojovým kódom (open-source).

Časť aplikácie, ktorá najviac ovplyvní kvalitu hudby (zvukového výstupu) je bezpochyby hudobné jadro. Jeho základom je hudobný CSP, zapísaný vo forme programu. Pre túto úlohu som zvolil aplikáciu Strasheela. Je to systém pre kompozíciu hudby na základe obmedzení a využíva programovací jazyk Oz [21]. Je dostupná vo všetkých operačných systémoch. Výhodou je výborná

dokumentácia a mnoho ukážkových vzorových príkladov, čím poskytuje viac priestoru pre modifikácie pre prípadných záujecov. Strasheela má potenciál splniť ďalšiu zo základných požiadaviek, a to generovanie hudby v reálnom čase. Pre tieto účely má zabudované rozhranie OSC (open sound control) [22]. OSC je primárne dostupné len pre UNIXové systémy, avšak objavujú sa aj možnosti, ako tento protokol využiť v prostredí Windows.

Zvyšok aplikácie bude naimplementovaný v objektovo orientovanom programovacom jazyku JAVA. Je široko rozšírený na všetkých operačných systémoch a platformách a nemá žiadne problémy s prenositeľnosťou. Navyše vo svojom zvukovom rozhraní priamo poskytuje podporu pre prácu s MIDI a aj implementáciu základných MIDI zariadení ako je sekvencer a syntetizér. Taktiež je voľne dostupná aj knižnica pre komunikáciu pomocou OSC protokolu.

Verím, že výberom týchto nástrojov bude navrhovaná aplikácia dostupná pre všetky hlavné operačné systémy (Windows, LINUX, MacOS).

4 Návrh systému

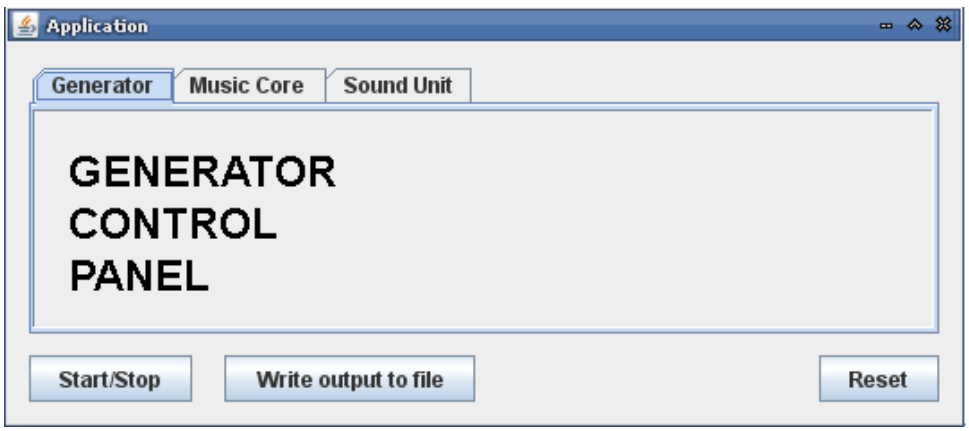
Kapitola o návrhu systému je na základe analýzy rozdelená na štyri celky: rozhranie, generátor, hudobné jadro a zvukovú jednotku. Ich návrh je ovplyvnený mojim výberom vstupných dát a systému pre riešenie CSP.

4.1 Rozhranie

Jedným z cieľov práce je návrh ľahko modifikovateľného systému. Pre tento účel má slúžiť rozhranie medzi základnými komponentmi a samotnou aplikáciou. V prípade potreby tak bude možné komponenty bez problémov nahradiť novou verziou alebo dokonca úplne novou implementáciou. V tejto kapitole je okrem toho popísaný aj návrh grafického uživateľského rozhrania (GUI). GUI je totiž neodmysliteľnou súčasťou každej desktopovej aplikácie.

Rozhranie medzi komponentmi a aplikáciou bude riešené pomocu abstraktných tried, z ktorých budú musieť komponenty dediť. Výhodou abstraktných tried je možnosť implementácie niektorých metód priamo v tejto triede. Zároveň stále môžme deklarovať metódy, ktoré musia byť implementované triedou, ktorá ich rozširuje. Medzi metódy implementované priamo v abstraktných triedach bude patriť napr. posielanie správ medzi komponentmi. Každý odvodený komponent bude musieť implementovať abstraktnú metódu pre získanie ovládacieho panelu, reštart funkcionality a pre ukončovaciu rutinu. Okrem toho generátor bude musieť implementovať metódy pre spustenie a prerušenie generovania a zvuková jednotka musí vedieť zapisovať výstup do MIDI súboru.

Náčrt uživateľského rozhrania aplikácie je zobrazený na obrázku 4.1. V hornej časti sa nachádzajú tri záložky. Každá karta predstavuje ovládací panel jedného komponentu a obsah závisí len na jeho implementácii. Spodná časť je vyhradená pre ovládacie prvky, ktoré musia byť vždy implementované.



Obrázok 4.1 – Návrh grafického uživateľského rozhrania aplikácie

Správy, ktoré si budú komponenty posielať, sa nazývajú správy rozhrania. Budú pozostávať z čísla udávajúceho poradie MIDI nôt (tzv. „tick“) a mapy dvojíc, ktoré charakterizujú priebeh na MIDI kanáloch. Jedna takáto dvojica bude tvorená číslom kanálu a zoznamom udalostí rozhrania. Udalosť rozhrania bude slúžiť ako určitá náhrada MIDI udalosti, ktorú poskytuje JAVA. Primárne bude určená pre posielanie udalostí o stlačení a uvoľnení klávesy (NOTE ON a NOTE OFF).

4.2 Generátor

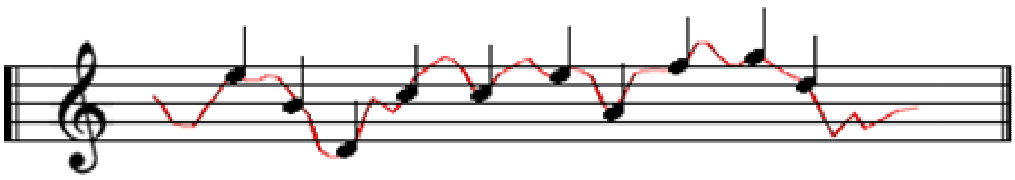
Úlohou generátora bude produkovať vstup pre hudobné jadro. Časovač bude v pravidelných intervaloch poskytovať podnety k odoslaniu správy rozhrania. Bude implementovaný pomocou triedy `javax.swing.Timer`. Či sa správa rozhrania odošle závisí na interpretácií vstupných dát agentom v danom časovom okamihu.

Súčasťou generátora musí byť aj ovládací panel. Jeho prvky umožnia načítavanie, modifikáciu, zobrazovanie vstupných dát a ovládanie agenta. Získavanie vstupných dát a návrh agenta je rozpísaný v ďalšom texte.

4.2.1 Vstupné dáta

Skôr než začneme rozoberať samotné spracovanie vstupných dát, musíme najprv definovať aké vstupné dáta sú prípustné. Z kapitoly 2.1 vieme, že ako vstup sa dá použiť takmer všetko (DNA, informácie zo svetovej burzy..). Vo väčšine prípadov sa však jedná o generátory pseudonáhodných čísel. Mojím nápadom je použiť ako vstupné dáta reálne informácie zo sveta okolo nás, zo sveta po ktorom chodíme. Terén poskytuje prekvapivo veľa zaujímavých a vhodných informácií pre generovanie hudby.

V najjednoduchšom prípade by stačilo vziať prierez výškovou mapou a „napasovať“ ho priamo do notovej osnovy ako na obrázku 4.2. Tento jednoduchý princíp má jednu značnú výhodu, poskytuje totiž spojitú informáciu, ktorá je dostatočne náhodná. Je dokonca priam nemožné zaznamenať všetky výškové prierezy krajiny, čo znamená značnú variabilitu. Výšková mapa okrem nadmorskej výšky poskytuje aj ďaľšie využiteľné informácie. Za zmienku určite stojí sklon terénu, jeho orientácia voči svetovým stranám, osvetlenie terénu slnkom alebo spádové krivky.



Obrázok 4.2 – Prierez terénom v notovej osnove

Nemusíme sa ale obmedzovať len na výškovú mapu. Využiť sa dajú aj mapy pôdnych typov, povodí, erózie, pokrytia územia flórou a faunou, ciest a železníc, inžinierskych sietí a mnohých ďalších. Záleží len na ich vhodnom interpretovaní. Pre jednu oblasť teda môžeme mať niekoľko rôznych druhov máp. Po vzore geografických informačných systémov (GIS) ich môžme organizovať do vrstiev. Z toho vychádza požiadavka na podporu rôznych formátov a načítavanie viacerých súborov. Jediným obmedzením na vstupné dáta je ich velkosť a to, že všetky načítané vrstvy musia mať rovnaké rozmery. Aplikácia nebude brať ohľad na to, či súhlasia geografické súradnice. To by zbytočne komplikovalo prípravnú fázu generovania a vo výsledku by to mohlo uživateľa odradiť. Pre čo najväčší komfort bude priamo v generátore možnosť vytvárať nové vrstvy, ktoré sa dajú odvodiť z výškovej mapy terénu. Jednotlivé vrstvy budú môcť byť načítané aj z klasických rastrových formátov pre ukladanie obrazových dát (*.jpeg, *.png, *.gif). Dôvodom je, že terénne informácie nebývaju tak ľahko dostupné, v niektorých prípadoch sú dokonca spoplatnené.

Samotné vstupné dáta sa ťažko interpretujú bez dodatočných informácií. Preto každá vrstva bude uchovávať aj metadáta, ktoré jej dodajú sémantiku. Mapové vrstvy vhodné pre GIS ich už majú väčšinou definované vo svojich formátoch, pre obyčajný obrázok ich musíme definovať my. K tomu poslúži jednoduchý panel alebo dialóg s výberom z predpripravených možností.

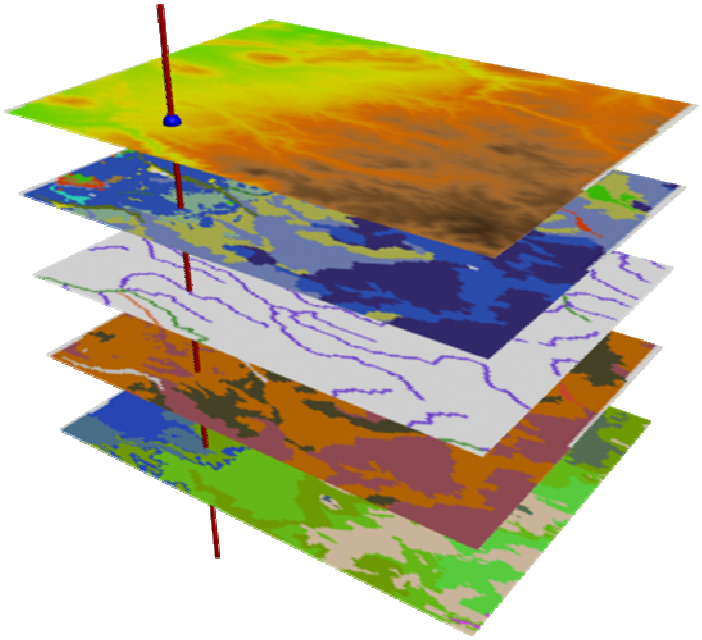
4.2.2 Agent

Keď sú všetky vstupné dáta pripravené, chýba už len spôsob, ako z nich získať relevantné informácie ku generovaniu hudby. K tomuto účelu bude použitý agent, ktorý sa „prechádza po teréne“.

V pravidelných intervaloch, stanovených časovačom generátora sa určí aktuálna pozícia agenta a zaznamenajú sa informácie z každej vrstvy v danom mieste (obrázok 4.3). Týmto spôsobom sa získa vektor hodnôt, ktorý je možné zanalyzovať. Agent predpokladá aspoň jednu zo základných vstupných vrstiev aby mohol generovať výstup. Za základné vrstvy sa považuje vrstva výšok, dĺžok a hlasitosti nôt.

Agent nemusí generovať iba noty. Jedna alebo viacro vrstiev môže byť priradených ku generovaniu kontrolných MIDI správ ako je zmena tempa, hudobného nástroja, či spúšťanie špeciálnych zvukových efektov.

Pre samotný pohyb agenta po vstupných dátach budú exisovať dve možnosti. Buď sa cesta zadá myšou, alebo sa nastaví tzv. „explore“ mód, v ktorom sa agent náhodne prechádza a objavuje svoje okolie, prípadne sleduje niektoré významné vlastnosti vstupných dát.



Obrázok 4.3 – Agent a vrstvy vstupných dát

4.3 Hudobné jadro

Tento komponent bude ako jediný príjemcom aj odosielateľom správ rozhrania. Je kľúčový z hľadiska kvality výsledného produktu. Jeho implementácia určí skoro všetky významné črty skladby. Po obdržaní správy rozhrania z generátora sa vyextrahujú jednotlivé udalosti rozhrania. Pre CSP program sú dôležité iba MIDI udalosti aktivácie tónu, tie majú v správe stanovenú aj hodnotu trvania (rôznu od nuly). Ostatné správy sa môžu priamo preposlať do zvukovej jednotky.

Ako už bolo naznačené, základ tvorí CSP program vytvorený pre systém Strasheela. Komunikácia so Strasheelou bude prebiehať pomocou OSC protokolu. Jadro bude mať implementovaný vstupný a výstupný OSC port, cez ktoré budú posielané OSC pakety. Podobne bude vyzerať aj CSP program, tiež musí mať vstupný aj výstupný OSC port. Aby komunikácia medzi aplikáciou a Strasheelou prebiehala, musí byť spustený OzServer. Beží na ňom kompilátor, ktorý očakáva ľubovoľný zdrojový Oz súbor. V našom prípade to bude CSP program, ktorý obsahuje OSC rozhranie, volania riešiteľa obmedzení a definície hudobných pravidiel. CSP program bude vytvorený za behu aplikácie, pred spustením samotného generovania. Bude možné ovplyvniť iba sadu hodobných pravidiel. K tomu poslúži ovládací panel hudobného jadra.

4.4 Zvuková jednotka

Zvuková jednotka bezprostredne po prijatí správy z hudobného jadra vytvorí príslušné MIDI udalosti, ktoré predá syntetizéru a sekvenceru. Syntetizér pre okamžité prehrávanie nepotrebuje informácie o čase, ale pre sekvencer je tento údaj veľmi dôležitý. Sekvencer zapisuje MIDI udalosti do stôp (tracks) a využíva túto informáciu pri ich prehrávaní. Nastavenie tempa a rôznych druhov hudobných nástrojov prebieha tiež pomocou prijatých udalostí. Pre obe MIDI zariadenia budú použité existujúce triedy `Synthesizer` a `Sequencer` odvodené z rozhrania `javax.sound.midi.MidiDevice`. Zvuková jednotka musí podporovať zápis do súboru, pre tento účel existuje metóda `MidiSystem.write()`. Do súboru sa týmto spôsobom zapíše sekvencia stôp zo sekvenceru.

Úlohou zvukovej jednotky bude aj vytváranie MIDI správ, ktoré ukončujú znenie tónu (NOTE OFF). K tomu sa použije údaj o dĺžke tónu. Pri každej udalosti NOTE ON sa vloží príslušná udalosť NOTE OFF do posuvného registra, na miesto určené dĺžkou tónu. V jednej bunke registra môže byť uložených viacero udalostí, aplikujú sa súčasne pri ich vysunutí.

K dispozícii môže byť naraz viacero softvérových alebo hardvérových MIDI zariadení a nie všetky musia byť v danom okamihu dostupné. Ovládací panel zvukovej jednotky bude slúžiť práve na výber zariadení, ktoré chceme použiť. Tie budú rozdelené do dvoch zoznamov, na syntetizéry a sekvencery.

5 Implementácia

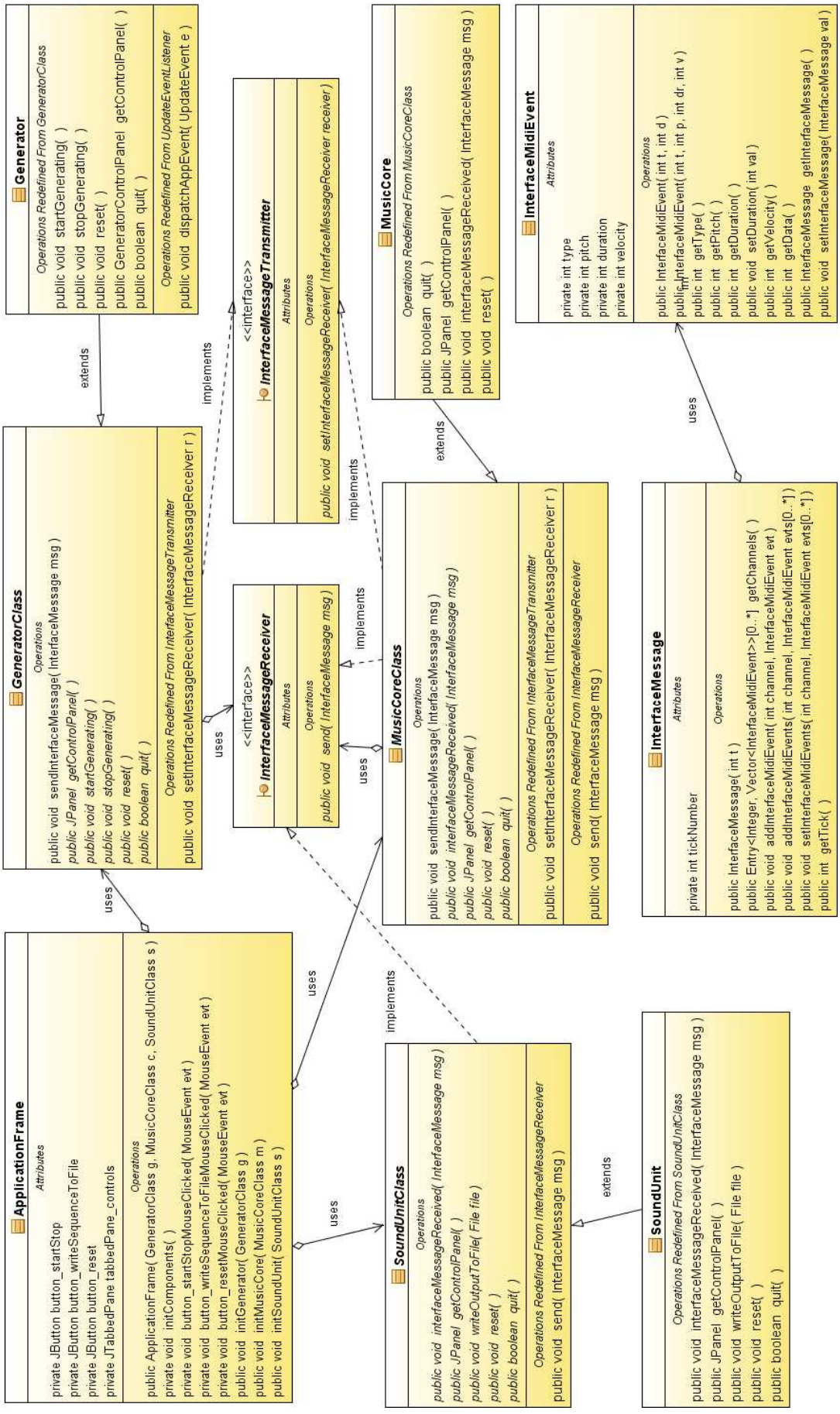
Po návrhu a analýze pristúpime k samotnej implementácii systému pre algoritmickú kompozíciu hudby. Kapitola je opäť rozčlenená do podkapitol pre rozhranie a jednotlivé komponenty generátora, hudobného jadra a zvukovej jednotky. Okrem detailného postupu implementácie sa vyskytne aj popis princípu niektorých použitých algoritmov a zbežná špecifikácia protokolu OSC. Spomeniem aj problémy, ktorým som čelil.

5.1 Rozhranie

Najlepšiu predstavu o implementácii rozhrania systému ponúka diagram tired znázornený na obrázku 5.1. Diagram obsahuje rozhranie pre posielanie (`InterfaceMessageTransmitter`), resp. prijímanie (`InterfaceMessageReceiver`) správ rozhrania. Tieto rozhrania sú stavebnými prvkami komunikačných jednotiek komponentov. Posielaním správ rozhrania medzi komponentmi sa rozumie posielanie objektov `InterfaceMessage`. Takáto správa má jednoznačný identifikátor v podobe časového údaju (atribút `tickNumber`). Jedna správa rozhrania môže obsahovať neobmedzený počet `InterfaceMidiEvent` udalostí. Udalosti rozhrania sú zoskupené podľa MIDI kanálu, na ktorom nastali a slúžia ako obálka pre prenášané dáta.

Komunikačné jednotky sú priamo implementované v abstraktných triedach generátora (`GeneratorClass`), hudobného jadra (`MusicCoreClass`) a hudobnej jednotky (`SoundUnitClass`). Výhodou je, že komponenty odvodené z týchto abstraktných tried už môžu využívať metódy komunikačných jednotiek a nezaťažovať sa ich implementáciou. Samotné komponenty musia teda implementovať iba niektoré metódy svojich abstraktných tried. Každý komponent musí vedieť pomocou metódy `getControlPanel()` vrátiť panel so svojimi ovládacími prvkami. Okrem toho musia všetky komponenty implementovať metódy `reset()` a `quit()`. Generátor musí navyše implementovať metódy `startGenerating()` a `stopGenerating()`, ktoré sa starajú o generovanie správ rozhrania pre hudobné jadro. Zvuková jednotka musí implementovať metódu `writeOutputToFile()`, ktorá zapíše zvukový výstup do súboru.

Aplikácia dokáže použiť ľubovolnú implementáciu komponentov vytvorených rozšírením svojich abstraktných tried. Pokladám to za veľkú výhodu a pridáva to systému na atraktivite. Môže vzniknúť niekoľko variácií každého komponentu a ich kombináciami sa určite dá dopracovať k neuveriteľnému počtu hudobných výstupov.



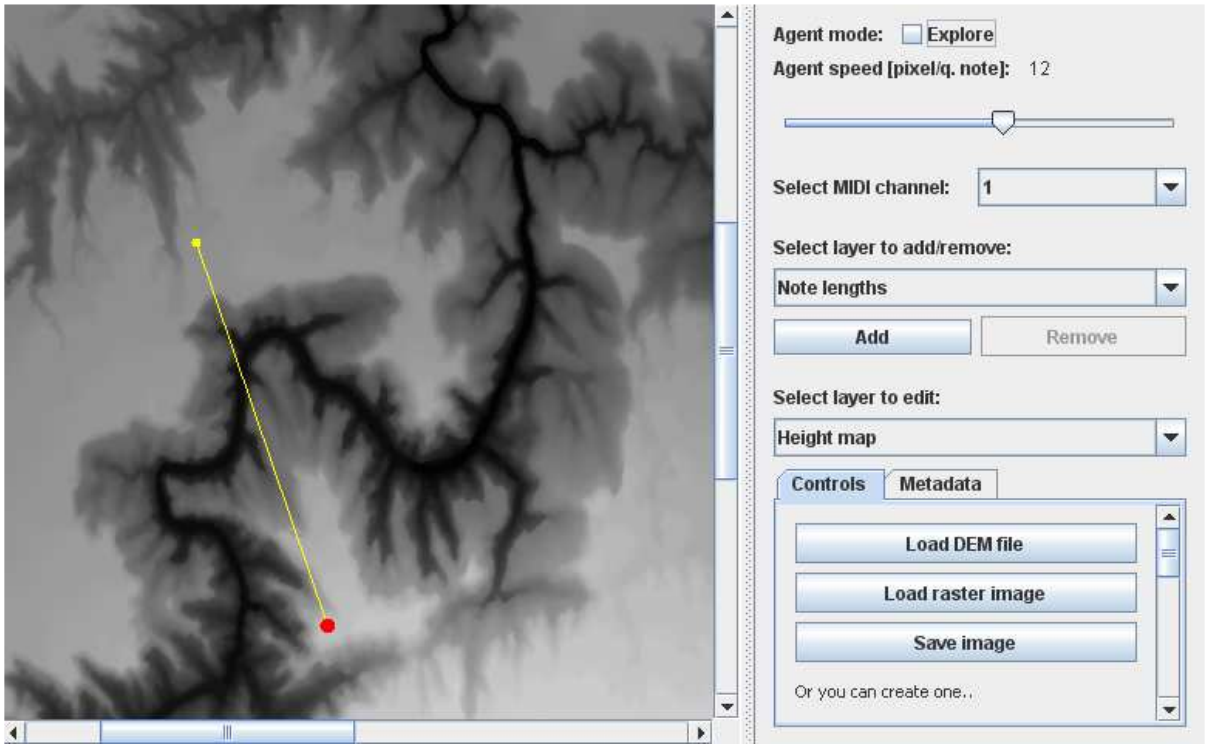
Obrázok 5.1 – Diagram tried rozhrania

5.2 Generátor

Generátor nôt je kľúčovým komponentom v mojom programe pre tvorbu hudby počítačom. Ostatné komponenty sú bez jeho výstupu nepoužiteľné a preto mu bola pri implementácii venovaná najväčšia pozornosť. V tejto kapitole si najprv priblížime ovládací panel generátoru, potom do hĺbky rozoberieme jednotlivé vrstvy a možnosti agenta. V závere je popísaný samotný postup generovania hudobného materiálu.

5.2.1 Ovládací panel

Ovládací panel generátoru tvoria dva samostatné celky (obrázok 5.2). V ľavej časti je zobrazovacia plocha, do ktorej sa vykresľujú rastrové obrázky uložené v jednotlivých vrstvách. V nasledujúcom texte bude rastrový obraz uložený vo vrstve označovaný ako mapa danej vrstvy. Plocha reaguje na 3 rôzne podnety a podľa ich typu sa prekresľuje. Najčastejším prípadom je zmena pozície agenta. Tu stačí vziať aktuálne zobrazenú mapu a vykresliť začiatočnú a koncovú pozíciu agenta spojenú úsečkou. Druhým podnetom je zmena vrstvy, prípadne MIDI kanálu. V oboch prípadoch sa zobrazí mapa aktuálne zvolenej vrstvy. V prípade, že mapa neexistuje, bude zobrazovacia plocha vymazaná. Posledným podnetom sú individuálne udalosti jednotlivých vrstiev, ktoré vyžadujú prekleslenie mapy.



Obrázok 5.2 – Ovládací panel generátoru

Pravá časť je vyhradená pre ovládacie prvky generátoru. Obsahuje nastavenia pre agenta, roletové menu pre výber MIDI kanálu a panel s ovládaním vrstiev zvoleného kanálu. Každý MIDI kanál má vlastné ovládanie vrstiev a pozostáva z roletového menu a tlačítok pre pridanie a odobranie vrstvy, a roletového menu pre výber jednej z pridaných vrstiev. V prípade, že bola pridaná aspoň jedna vrstva je zobrazený aj ovládací panel danej vrstvy.

5.2.2 Vrstvy máp

V rámci tejto práce boli implementované tri vrstvy pre základné vlastnosti tónu – výška, dĺžka a hlasitosť. Tie sú prístupné z každého MIDI kanálu a je tak možné definovať až 16 rôznych vstupov výšok, dĺžok a hlasitostí.

Každá vrstva umožňuje načítavanie a ukladanie rastrových obrázkov, má vlastný generátor máp a metadáta. Metadáta mapám poskytujú dodatočné informácie potrebné ku generovaniu. Všetky používané vrstvy musia mať rovnakú veľkosť, v opačnom prípade je zobrazené chybové hlásenie a vstup je odmietnutý. Rozoberme si teraz jednotlivé vrstvy podrobnejšie.

5.2.2.1 Vrstva výšok nôt

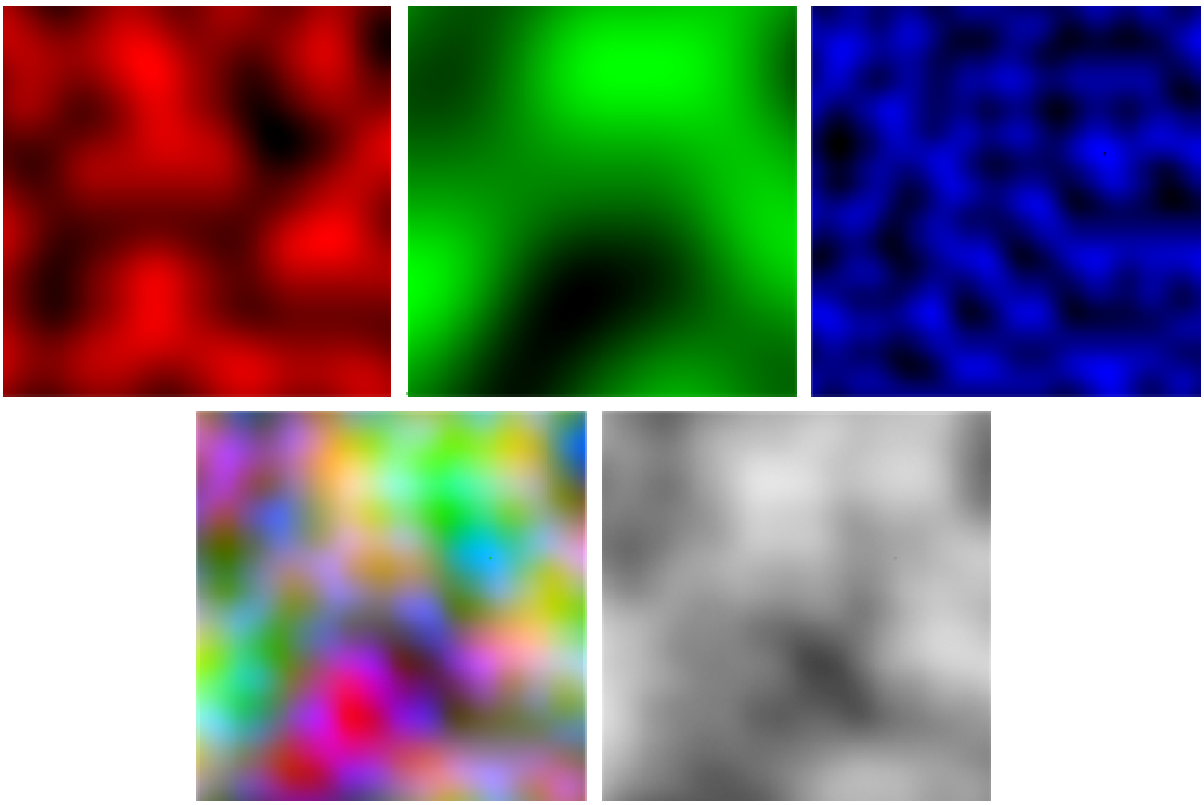
Najvýznamnejšou vlastnosťou tónu je bezpochyby jeho výška. Bez jej variability by sme nemohli skladať zaujímavé melódie a ani vytvárať akordy. Mapa tejto vrstvy je navyše vstupom pre generátory máp dĺžky a hlasitosti nôt.

Ústrednou myšlienkou tejto práce je využite terénnych informácií. Nadmorská výška je vlastnosť terénu, ktorá sa priam ponúka k určovaniu výšky tónu. Okrem priamočiareho načítania rastrovej výškovej mapy terénu je možné dáta získať z formátu DEM (digital elevation model). Tento formát bol vyvynutý inštitúciou USGS (United States Geological Survey) pre ukladanie rastrovo založených digitálnych výškových máp v textovej podobe. Napriek tomu že ide o zastaralý štandard je doteraz široko používaný a podporovaný väčšinou GIS. Triedy pre načítavanie DEM súborov vytvoril Dr. Mark O. Pendergast, Phd. vrámci jedného zo svojich projektov. Načítané dáta následne prevádzam do obrázku v 256 úrovniach šedi (v ďalšom texte označované už len ako úrovne šedi).

Metadáta výškovej mapy obsahujú nastavenia ako minimálna a maximálna výška tónu a metóda určovania jeho výšky. Prípustné hodnoty pre výšku tónu sú 0-127 (od C-1 do G9), čo je v súlade zo štandardom MIDI. Sú implementované dve metódy určovania výšky tónu. Prvá je priame mapovanie nadmorskej výšky do zadaného rozsahu výšky noty. V druhej metóde sa za výšku tónu pokladá nameraná nadmorská výška modulo rozsah výšok nôt. Obe metódy majú svoje výhody a záleží na vstupných dátach a užívateľovi, pre ktorú sa rozhodne. Okrem toho je možné nastaviť, ktoré farebné kanály sa z mapy majú pri generovaní použiť. Pre generovanie melódie nám stačí jeden farebný kanál – červený (R), zelený (G) alebo modrý (B). Môžeme však použiť aj všetky tri súčasne a vypočítať z nich jednu hodnotu – RGB (R+B+G/3) alebo intenzitu farby (0,30R+0,59G+0,11B). Pri určovaní intenzity farby v obraze ide v podstate o prevod obrazu do úrovní šedi. Koeficienty sú

určené stavbou ľudského oka, ktoré je najviac citlivé na zelenú a najmenej na modrú farbu. Ak ale chceme z tejto vrstvy generovať dvojice tónov alebo akordy (tri tóny), používame kombinácie farebných kanálov. Keďže výšková mapa vytvorená z DEM súboru je prevedená do obrazu v úrovniach šedi, nie je vhodným kandidátom pre tvorbu takýcho harmonických súzvukov (R, G aj B kanál má rovnakú hodnotu). Naopak vhodným vstupom sú dáta s veľkou variabilitou hodnôt medzi jednotlivými kanálmi, ako sú napr. fotky.

Poslednou možnosťou ako získať mapu výšok nôt je pomocou vstavaného generátoru. Ten dokáže vytvoriť mapy od 200x200 do 2000x2000 pixelov. Je založený na 3D Perlinovom šume, ktorý je v rôznych variáciách v počítačovej grafike široko používaný. Využitie našiel napr. v oblastiach ako je simulácia oblakov, ohňa, mramoru alebo vodnej hladiny a krajiny. Práve posledná z uvedených možností je základom pre tento generátor. Implementácia pochádza od samotného Kena Perlina, tak ako ju popísal vo svojom článku [23]. Perlinov šum je špecifický hlavne tým, že pre konkrétne súradnice v priestore je vypočítaná vždy rovnaká hodnota. Použiť 3D Perlinov šum pre vygenerovanie 2D výškovej mapy sa može zdať prehnané, ale opak je pravdou. Z-súradnicu môžeme použiť ako časovú os a získať tak oveľa väčší priestor máp. Okrem času sa dá nastaviť aj frekvencia šumu, tá ovplyvňuje počet (veľkosť) vytvorených „kopcov a údolí“. Pre dosiahnutie realistického efektu je zvykom priemerovať viacero výstupov s rôznou frekvenciou. V generátore je možné pre každý farebný kanál, ktorý sa má použiť pre generovanie nôt, samostatne definovať vlastnosti Perlinovho šumu. Na obrázku 5.3 sú zobrazené vytvorené výškové mapy s rôznou frekvenciou a časom pre každý farebný kanál a ich spojenie do RGB a úrovní šedi.



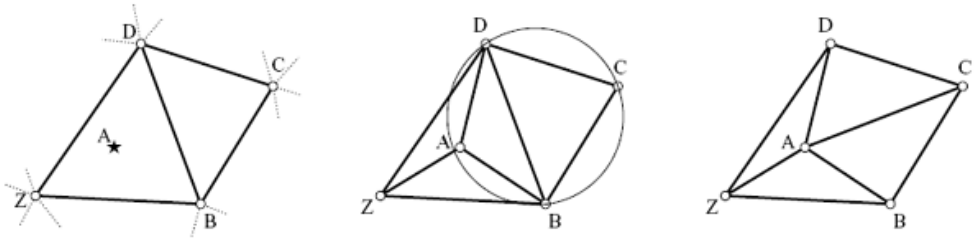
Obrázok 5.3 – Ukážka výstupu generátoru máp výškok nôt

5.2.2.2 Vrstva dĺžky nôt

Rytmus spolu s melódiou a harmóniou tvoria tri základné prvky hudobného prejavu. Jeden zo spôsobov ako ho dosiahnuť je striedanie dlhších a kratších tónov. Úlohou tejto vrstvy je pomocou mapy definovať jednotlivé dĺžky nôt a simulovať tak rytmus.

Každá farba mapy má v metadátach priradenú určitú dĺžku, pričom okrem nôt je možné farbu priradiť aj pomlčkám. Celkovo je možné zvoliť šesnástinovú až celú notu, vrátane nôt s bodkou, ktoré predlžujú dobu trvania o polovicu. Pomlčky sú v rovnakom rozsahu, od šestnástinovej po celú. Pri načítaní rastrového obrázku sa overuje počet farieb, ak presiahne počet možných hodnôt je zobrazená chybová hláška a vstup je odmietnutý.

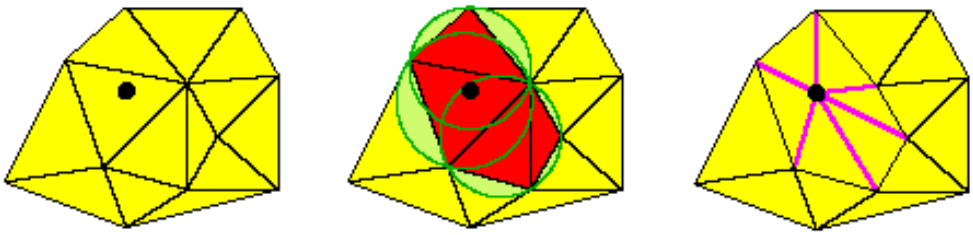
Ak je pre daný MIDI kanál načítaná aj výšková mapa, je možné použiť generátor a vytvoriť z nej mapu dĺžok nôt. Generátor máp využíva adaptívnu segmentáciu obrazu na trojuholníkovú sieť. Pre tento účel je implementovaný inkrementálny algoritmus Delaunayho triangulácie, ktorý navrhli Lawson a Sloan [24]. Základnom Delaunayho triangulácie je test pomocou kružnice opísanej vrcholmi trojuholníka. Po vložení nového bodu do triangulácie sa overí, či je vo vnútri opísanej kružnice nejakého trojuholníka. Ak áno, aplikuje sa princíp maximalizácie minimálnych uhlov (obrázok 5.4). Tento príncíp je vhodným riešením triangulácie štvoruholníka. Vyberie sa tá diagonála, ktorá maximalizuje minimum všetkých šiestich vnútorných uhlov dvoch trojuholníkov. Veľkou výhodou je, že zo zadanej množiny bodov existuje len jedna, a tým pádom unikátna, Delaunay triangulácia. Navyše externé hrany triangulácie vytvárajú konvexnú obálku vložených bodov.



Obrázok 5.4 – Test pomocou kružnice

Približný postup použitého inkrementálneho algoritmu je:

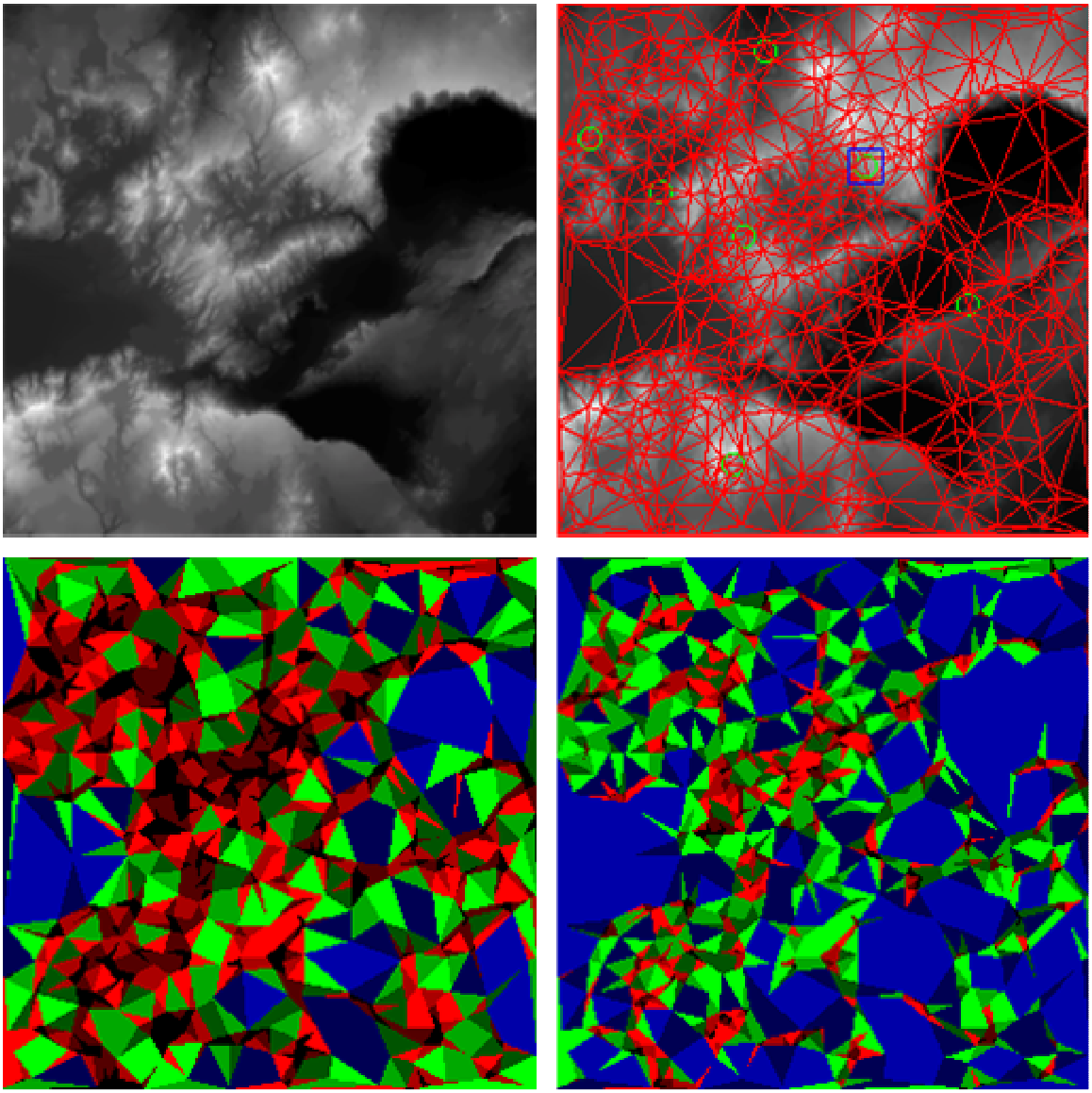
1) vytvor externý trojuholník, ktorý obsahuje všetky uvažované vkladané body
2) body sa vkladajú po jednom, každé vloženie spôsobí sériu zmien v triangulácii (obrázok 5.5)
   a) vlož bod
   b) nájdi všetky trojuholníky, ktorých opísaná kružnica obsahuje vkladaný bod
   c) vymaž ich z triangulácie - vytvorí konvexnú oblasť
   d) z vkladaného bodu spusť hrany do každého bodu konvexnej oblasti
   e) opakuj bod 2) kým existuje bod na vloženie



Obrázok 5.5 – Ilustrácia inkrementálneho algoritmu

Body sa do siete vkladajú podľa určitých kritérií. Pre každý trojuholník triangulácie sa vypočíta a uloží kandidát, teda bod ležiaci v trojuholníku, ktorý má maximálnu odchýlku od roviny trojuholníka. Rovinou trojuholníka sa rozumie rovina prechádzajúca jeho vrcholmi, kde z-súradnica vrcholov je intenzita farby pixelu v tomto bode. Z vypočítaných kandidátov sa za najlepšieho označí ten, ktorý patrí trojuholníku s najväčším obsahom. Tento bod sa v ďalšej iterácii vloží do siete, vzniknú tak nové trojuholníky, pre ktoré sa vypočíta kandidát a zopakuje sa výpočet najlepšieho kandidáta. Výpočet adaptívnej segmentácie sa zastaví po dosiahnutí zadaného maximálneho počtu vložených bodov alebo ak už pre žiadny trojuholník neexistuje kandidát. Trojuholník siete nemá kandidáta, ak všetky jeho body vykazujú odchýlku od roviny trojuholníka menšiu ako je zvolená tolerancia alebo ak sú všetky jeho pixely „použité“. Za použité sa považujú pixely v určitom okolí od vloženého bodu, pričom veľkosť okolia závisí na vstupnom parametri. Všetky tri parametre sa nastavujú v ovládacom paneli generátoru mapy.

Výsledkom generátora je rastrový obraz, ktorý vznikne vyfarbením trojuholníkovej siete podľa jednej z dvoch metód, ktorú je možné zvoliť v metadátach. Prvá je založená na veľkosti trojuholníkov. Postupuje sa od najmenšieho trojuholníka a každej farbe je pridelená takmer rovnaká časť povrchu. Druhou metódou je vyfarbovanie trojuholníkov podľa času, kedy boli vytvorené, bez ohľadu na ich velkosť. Na obrázku 5.6 je v hornom rade znázornený vstup do generátoru (výšková mapa oblasti Klamath Falls, Oregon, USA) a priebeh výpočtu trojuholníkovej siete. V dolnom rade sa nachádzajú výstupy pre metódu vyfarbovania podľa veľkosti a veku trojuholníka. Výhodou je, že metódu môžeme meniť aj po dokončení výpočtu. Okrem toho môžeme nastaviť aj počet farieb, ktoré majú mať priradenú hodnotu dĺžky noty (táto funkcionalita nie je dostupná po načítaní rastrového obrázku).



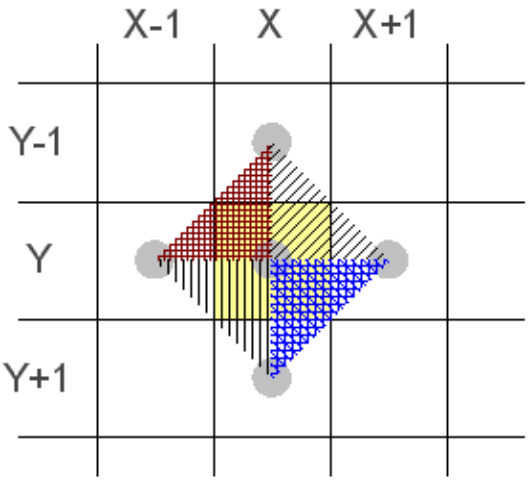
Obrázok 5.6 – Ukážka výstupu generátoru máp dĺžok nôt

5.2.2.3 Vrstva hlasitosti nôt

Poslednou implementovanou vrstvou je vrstva hlasitosti a jej úloha je jednoduchá a priamočiara. Jej mapa určuje či, a ako veľmi, bude noty generované pre daný MIDI kanál počuť. Vrstva sa teda dá použiť na dočasné „vypnutie“ MIDI kanálu bez nutnosti odstránenia ostatných vrstiev. V metadátach vrstvy sa dá nastaviť minimálna a maximálna hlasitosť v rozsahu 0-127 (rozsah korešponduje so štandartom MIDI).

Tak ako predošlé vrstvy, aj táto má implementovaný vlastný generátor máp. Opäť je však nutné načítať mapu výšok nôt, pretože sa používa ako základ pre generovanie mapy hlasitosti. Princíp generátoru máp je odvodený z reálneho prostredia a simuluje osvetlenie terénu slnkom. Aby sme tento efekt dosiahli, musíme poznať samotný terén a smer odkiaľ dopadajú slnečné lúče. Terén dokážeme odvodiť z výškovej mapy, musíme však ešte v každom bode (pixely mapy) vypočítať normálu povrchu. Tú získame priemerom normál zo štyroch trojuhloníkov, ktoré sú vytvorené

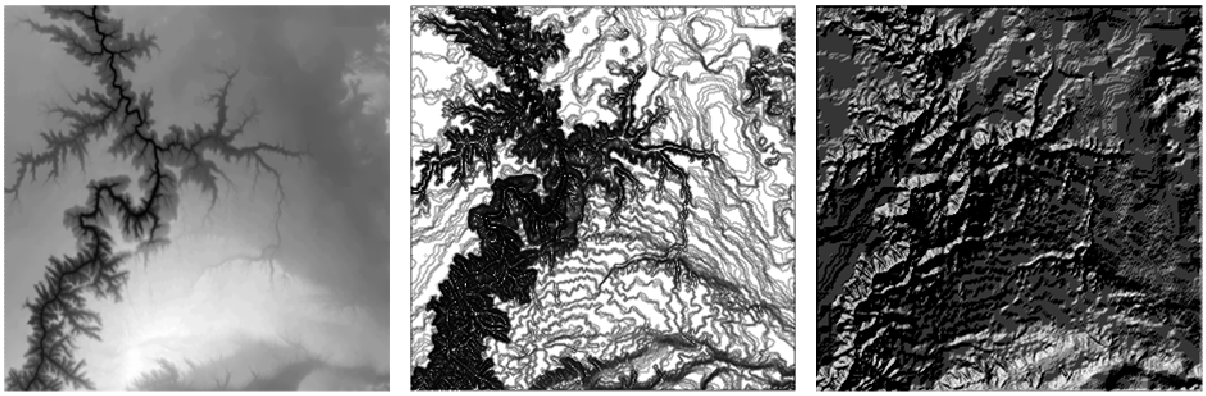
pomocou susedných bodov. Každý bod pomocného trojuholníka má x, y súradnicu (pozícia v rastry) a z-súradnicu, ktorá je odvodená z intenzity farby na danom mieste a predstavuje výšku (obrázok 5.7).



Obrázok 5.7 – Pomocné trojuholníky pre výpočet normály v bode [x, y]

Pri výpočte Slnko považujem za dostatočne vzdialený svetelný zdroj, ktorého lúče sú pri dopade na povrch navzájom rovnobežné. Týmto spôsobom stačí vypočítať jeden vektor, ktorý je spoločný pre všetky dopadajúce lúče. K jeho určeniu sú potrebné dva údaje, a to smer z ktorého lúč dopadá (0˚-359˚) a uhol medzi kolmicou na rovinu xy a dopadajúcim lúčom (0˚-90˚). Oba parametre sa dajú nastaviť na ovládacom paneli generátoru.

Výsledná farba pixelu v mape hlasitosti je vypočítaná ako veľkosť uhlu medzi dopadajúcim lúčom a normálou v danom bode. Ak uhol presiahne 90˚ tak to znamená, že pixel je „osvetlený“ zo spodnej strany. Táto situácia je neprípustná a pixel bude mať vo výslednej mape čiernu farbu. Naopak ak je uhol veľmi blízky 0˚ tak to znamená, že lúč na plochu dopadá kolmo a pixelu sa priradí biela farba. V ostatných prípadoch výsledná farba korešponduje s velkosťou tohoto uhlu (úrovne šedi). Medzi výslednou mapou a reálnou mapou osvetlenia terénu Slnkom je však jeden výrazný rozdiel. Implementácia neberie do úvahy možné prekážky, ktoré by pre daný bod zabránili dopadu lúča (napr. vysoké pohoria). Zisťovanie kolízii je totiž výpočetne náročný a zdĺhavý proces. Napriek tomu sú výsledky dostatočné a stále vhodné pre mapu hlasitosti nôt. Obrázok 5.8 demonštruje možné výstupy na základe výškovej mapy národného parku Grand Canyon (prvý zľava).



Obrázok 5.8 – Ukážka výstupu generátoru máp hlasitosti nôt

5.2.3 Agent

Ak je v ľubovoľnej vrstve načítaná mapa, môžeme na ňu umiestniť agenta. Ľavým tlačítkom myši sa nastavuje východzia poloha a pravým cieľová poloha agenta. Po spustení generovania sa agent začne pohybovať do miesta určenia. Prípadne ak chýba informácia o cieľovej pozícii a je zapnutý prieskumný mód (explore), agent so 70 percentnou pravdepodobnosťou pokračuje v smere, ktorým sa predtým pohyboval. Zvyšných 30 percet je vyhradených pre otočenie agenta o 10˚, s rovnakou pravdepodobnosťou vľavo alebo v pravo. Ak sa agent priblíži k okraju mapy, zatáča v smere kde musí spraviť menej otočení. Ak sa dostane do rohu, otočí sa o 180˚. Dôležitým nastavením je aj rýchlosť pohybu agenta. Tá sa zadáva ako vzdialenosť v pixeloch za štvrťovú notu a je ju možné meniť aj počas generovania.

5.2.4 Priebeh generovania

Najdôležitejšou funkcionalitou generátoru je, samozrejme, samotné generovanie nôt a ich posielanie do hudobného jadra. Generovanie je možné spusiť iba v prípade, že je agent umiestnený na mape, čo zároveň znamená, že musí byť načítaná aspoň jedna mapa. Nezáleží na tom aká to bude, pri absencii niektorej z máp sa pre daný MIDI kanál použijú základné nastavenia. Pre výšku tónu je to tón C4 (ekvilentné s MIDI notou s hodnotou 60), základnou dĺžkou je štvrťová nota a hlasitosť je nastavená na hodnotu 70.

Generátor ako jediný obsahuje časovač, ktorý dáva v pravidelných intervaloch agentovi podnet pre odoslanie správy rozhrania a následný pohyb vpred. Interval je určený na 125ms a korešponduje s dobou trvania šesnástinovej noty. Je vypočítaný podľa vzorca:

$$tickSize = \frac{60}{PPQ * BPM}$$

kde PPQ je počet úderov za štvťovú notu (v našom prípade 4 údery pre 4 šestnástinové noty) a BPM je počet štvrťových nôt za minútu (120 BPM predstavuje tempo Allegro moderato). Tieto parametre sú momentálne strikne dané, do budúcna sa však ráta s možnosťou ich nastavenia.

Správa rozhrania má štruktúru popísanú v kapitole 4.1 a odosiela sa aj v prípade, že neobsahuje žiadne udalosti rozhrania. Takáto „prázdna“ správa sa nespracováva v hudobnom jadre, ale je dôležitá pre zvukovú jednotku. Poskytuje jej informácie o čase, ktoré sú potrebné k správnemu prehrávaniu. S úderom hodín sa pre každý MIDI kanál zistí, či má načítanú aspoň jednu vrstvu. Ak áno, overí sa ešte, či na ňom práve neznejú nejaké tóny. Iba v tomto prípade sa extrahujú informácie z vrstiev MIDI kanálu, a to v mieste, kde sa agent práve nachádza. Ako už bolo spomenuté, ak niektorá z vrstiev chýba, použijú sa základné nastavenia. Udalosť rozhrania s typom NOTE ON sa do správy rozhrania pridá iba v prípade, že doba trvania a hlasitosť vygenerovanej noty je väčšia ako 0. Pomlčky túto podmienku nespĺňajú, pretože majú zápornú dobu trvania a nebudú tak nikdy súčasťou správy rozhrania. Ich úlohou je však blokovanie MIDI kanálu po dobu absolútnej hodnoty ich trvania. Pre tento účel je vytvorená pamäť, do ktorej je nota alebo pomlčka pridaná. Do pamäte sa vkladá iba číslo MIDI kanálu a absolútna doba trvania. S každým úderom hodín sa doba trvania zníži o 1, až kým nie je možné kanál uvoľniť a vygenerovať novú notu alebo pomlčku.

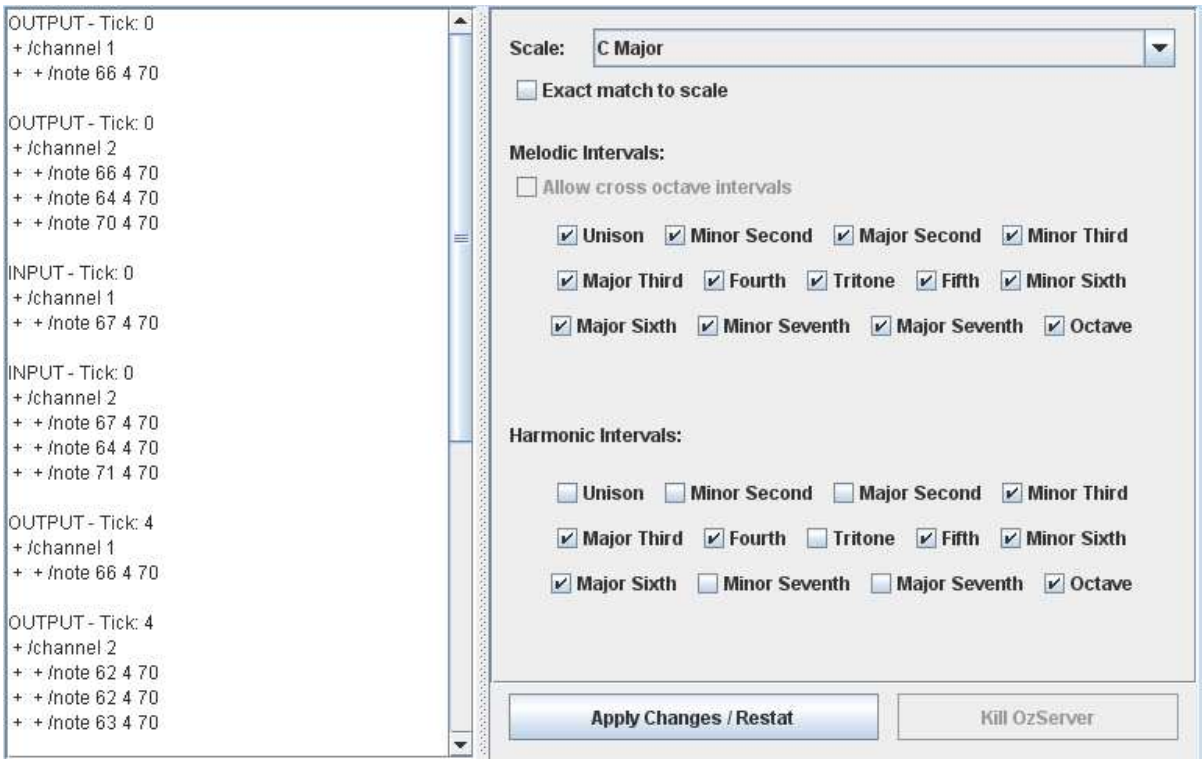
Generátor nemá za úlohu posielať aj správy o uvoľnení tónu (NOTE OFF). Dôvodom je odľahčenie spracovávania správ rozhrania v hudobnom jadre, navyše sú pre jeho beh irelevantné. Vytvorenie a spracovanie správ o uvoľnení tónu je ponechané na zvukovú jednotku.

5.3 Hudobné jadro

Hudobné jadro sa snaží previesť náhodný notový výstup z generátoru do „hudobnejšej“ podoby definovanej užívateľom. Najprv v skratke popíšeme užívateľské rozhranie a následne rozoberieme riešenie premeny hudobného materiálu pomocou problémov s obmedzujúcimi podmienkami (ďalej opäť označené len ako CSP). Musíme popísať, kde sa budú spracovávať, ako ich budeme riešiť, ako budeme prenášať dáta a aký formát dát potrebujeme.

5.3.1 Ovládací panel

Ovládací panel je vertikálne rozdelený na dva celky a je zobrazený na obrázku 5.9. V ľavej časti sa nachádza textová oblasť, do ktorej sa vypisujú odosielané a prijímané správy OSC (Open Sound Control) protokolu. Do oblasti sa okrem týchto správ vypisujú aj niektoré chybové hlásenia hudobného jadra (napr. chyba pri štarte OzServeru). V pravej časti je sada ovládacích prvkov, ktoré ovpyvňujú program vykonávaný na strane OzServeru a ich význam je vysvetlený v kapitole o hudobných pravidlách. V pravom dolnom rohu sú umiestnené tlačidlá pre štart (prípadne reštart) a vypnutie OzServeru.



Obrázok 5.9 – Ovládací panel hudobného jadra

5.3.2 OzServer

V kapitole 3.3 som si ako nástroj pre riešenie CSP zvolil program Strasheela. Jeho súčasťou je aj aplikácia OzServer, ktorá spustí plnohodnotný Oz kompilátor. Týmto spôsobom je možné spustiť Oz kompilátor z ľubovoľnej aplikácie a posielať z nej na načúvací port serveru Oz program, ktorý sa má vykonať. Druhým spôsobom, ktorý využíva aj moja aplikácia, je predať pri spúšťaní OzServeru celý súbor s požadovaným Oz kódom.

V oboch prípadoch je však podľa štandardu najprv načítaný konfiguračný súbor Oz kompilátoru OZRC. Ten bol vytvorený podľa návodu z webových strátok Strasheely. Obsahuje nastavenia ako sú cesty k potrebným aplikáciám a definície premenných nutných pre beh Strasheely. Konfiguračný súbor je súčasťou inštalačného balíku aplikácie, ale je nutné ho upraviť, prípadne jeho obsah pridať do už existujúceho OZRC súboru.

OzServer sa automaticky spúšťa pri štarte aplikácie a predpokladá sa, že cesta k nemu je pridaná do systémovej premennej PATH. V prípade neúspešného spustenia serveru aplikácia zobrazí chybové hlásenie a hudobné jadro iba prepošle výstup z generátoru do zvukovej jednotky.

5.3.3 Oz program

Pri štarte OzServeru sa okrem konfikuračného súboru OZRC načítava aj Oz program (v ďalšom texte nazývaný už len program), ktorý má v reálnom čase prijať, spracovať a odoslať OSC správy. Program

sa skladá z troch častí. Dynamická časť, ktorá je vygenerovaná pri spúšťaní OzServeru je vsunutá medzi dve statické časti.

5.3.3.1 Inicializácia a metódy spracovania vstupu

Prvá statická časť obsahuje program, ktorý najprv načíta moduly Strasheely pre prácu s OSC protokolom (funktor OSC) a pre riešenie CSP v reálnom čase (funktor RT). Potom sú zadefinované procedúry pre spracovanie vstupu.

Hlavnou procedúrov je `GetSolution()`, ktorá prijíma číslo MIDI kanálu a zoznam nôt obdržaných z aplikácie pre tento kanál. Ako napovedá názov procedúry, výsledkom je nájdené riešenie. V závislosti od počtu nôt sa pripraví zoznam možných vstupov. Napr. ak sú na vstupe 3 noty, hľadáme akord a zoznam možných vstupov má podobu:

[[1 2 3] [1 2] [2 3] [1 3] [1] [2] [3]]

kde jednotlivé čísla predstavujú rôzne objekty nôt s určitou výškou, dĺžkou a hlasitosťou. Hranaté zátvorky („[“ a „]“) sa v jazyku Oz používajú pre uzavreté zoznamy. Takéto zoznamy majú nemenný počet prvkov.

Procedúra `NextSolution()` postupne prechádza zoznam možných vstupov a vráti prvý pozitívny výsledok. Uvažujme príklad uvedený vyššie. Ak neexistuje riešenie pre akord [1 2 3], pristúpi sa k hľadaniu pre všetky dvojice, pričom sa ako prvá vezme dvojica nôt [1 2]. Ak ani jedna z dvojíc nemá riešenie, pokračuje sa jednotlivými notami. V najhoršom prípade je výsledkom prázdne riešenie.

Samotné získavanie výsledku má na starosti procedúra `CallSolver()`, ktorá vstupné noty transformuje do formátu vhodného pre Straseehlu (objekt `Score`) a podľa ich počtu zavolá príslušný riešiteľ obmedzení pracujúci v reálnom čase (realtime capeable constraint solver).

Okrem toho obsahuje táto časť programu procedúry pre zistenie intervalu medzi dvoma notami (vzdialenosť je udaná v poltónoch) a prevod noty z OSC do Strasheela reprezentácie a späť.

5.3.3.2 Dynamické vlastnosti a hudobné pravidlá

Pod pojmom dynamická časť sa myslí program vygenerovaný pri štarte OzServeru. V podstate ide o určitú šablonu, ktorá je doplnená informáciami z aplikácie. Ak chceme zmeniť vstupný alebo výstupný port, prípadne hodnotu niektorej z hudobných konštánt (PPQ, BPM), stačí zmeniť nastavenie iba v aplikácii.

Základným pravidlom je obmedzenie noty do kontrétnej stupnice `NoteFromScale()`. Aby sme niečo také mohli docieliť, musíme poznať noty, ktoré patria do zadanej stupnice. Tento údaj je predaný aplikáciou v podobe zoznamu siedmich hodnôt, ktoré predstavujú jednotlivé tóny. Hodnoty tónov sa pohybujú v rozsahu 0-11, čiže 12 poltónov, čo zodpovedá jednej oktáve. Tieto hodnoty získame ako číslo daného tónu v MIDI reprezentácii modulo 12. Rovnaký princíp sa potom používa

aj pri obmedzovaní množiny riešení. Okrem toho máme možnosť zvoliť, či vygenerovaná nota už musí byť v tejto stupnici alebo ju môžeme o pol tóna upraviť. Tento efekt dosiahneme pravidlom `RestrictInterval()`. Pravidlo udáva, že velkosť intervalu medzi vygenerovanou notou a hľadaným riešením je buď nulová (priama zhoda) alebo maximálne 1 (nepriama zhoda). K výberu stupnice a nastaveniu príznaku exaktnej zhody slúžia prvky umiestnené na ovládacom paneli hudobného jadra.

Uveďme si príklad riešenia jednoduchého hudobného CSP. Máme zadanú stupnicu C dur, definovanú ako [0 2 4 5 7 9 11] (čísla udávajú vzdialenosť v poltónoch od noty C). Dovolíme nepriamu zhodu a na vstupe je vygenerovaná MIDI nota s hodnotou 63 (D#4). Pred použitím obmedzujúcich podmienok môže byť riešením ľubovoľná nota z rozsahu 0-127 (obor hodnôt hľadaného riešenia teda na začiatku obsahuje všetky povolené hodnoty pre MIDI noty). Aplikujme najprv pravidlo nepriamej zhody, ktorým obmedzíme obor hodnôt riešenia na hodnoty, ktorých vzdialenosť od vstupnej noty je najviac 1 poltón. Z celého rozsahu teraz zostanú už iba MIDI noty 62 (D4), 63 a 64 (E4). Po aplikovaní podmienky stupnice sa rozsah prípustných hodnôt ešte zmenší a bude obsahovať už len hodnoty, ktoré po modulo 12 spadajú do definície stupnice. Napr. pre notu 62 je zvyšok po celočíselnom delení 2 a tým pádom nota spadá do stupnice, ktorú sme zadefinovali na začiatku. Existujú teda dve riešenia tohto CSP a to noty 62 a 64. Program samotný netuší, koľko riešení pre tento CSP existuje, vráti prvé na ktoré narazí. Princíp hľadania a výberu riešenia bol načrtnutý v kapitole 2.3. Ak by sme namiesto nepriamej zhody požadovali zhodu priamu, riešenie by neexistovalo, pretože žiadna z prípustných hodnôt riešenia by nesplnila obe zadané podmienky (nota 63 nie je súčasťou stupnice C dur).

Ak by sme zopakovali postup z uvedeného príkladu na vstupe s MIDI notou 64, dostaneme pre nepriamu zhodu tiež dve riešenia – 64 a 65 (F4) a jedno riešenie pre praimu zhodu – 64. Opäť musím podotknúť, že program nevie počet správnych riešení a preto môže byť pri nepriamej zhode za výsledok označená nota 65, napriek tomu, že vygenerovaná nota už patrí do zadanej stupnice.

Ďalším pravidlom, ktoré môžeme pri definícii CSP použiť, je `MelodicInterval()` (melodický interval). K tomu potrebujeme zoznam povolených melodických intervalov, ktorý získame pri vytváraní programu zo zaškrtávacích políčok na ovládacom paneli hudobného jadra. Základné nastavenie povoľuje všetky možné intervaly, ale iba v rámci jednej oktávy. To znamená, že v melódii sa nikdy neobjaví skok o viac ako oktávu. Samotné pravidlo obmedzí množinu riešení tak, že vzdialenosť medzi dvoma tónmi spadá do jedného zo zadaných melodických intervalov.

Posledným implementovaným pravidlom je `HarmonicInterval()` (harmonický interval). Povolené harmonické intervaly sa získajú rovnakým spôsobom, ako v prípade melodických intervalov. Za harmonické intervaly aplikácia od začiatku považuje malú a veľkú terciu, kvartu, kvintu, malú a veľkú sextu a oktávu. Je však možné vyberať zo všetkých možných intervalov v rámci oktávy. Pravidlo obmedzí množinu riešení tak, že vzdialenosť medzi dvoma tónmi spadá do jedného zo zadaných harmonických intervalov.

5.3.3.3 CSP a hľadanie riešenia

Druhá statická časť definuje procedúru reagujúcu na OSC správy, prehľadávače a procedúry s definíciou CSP (skripty). Pod pojmom prehľadávač sa myslí trieda Strasheely `ScoreSearcherWithTimeout`, ktorej je pri vytvorení inštancie predaný skript, maximálna doba hľadania a predvolené riešenie. Predvolené riešenie je výsledkom prehľadávania v prípade, že vyprší časový limit na nájdenie riešenia. Maximálna doba hľadania je stanovená na polovicu úderu hodín generátoru a predvoleným riešením je prázdne riešenie.

V procedúre, ktorá sa spustí pri prijatí OSC správy, sa najprv overí, či je prijatá OSC správa v očakávanom formáte. Ak áno, zavolá procedúru `GetSolution()` popísanú vyššie. Po obdržaní riešenia sa postará aj o jeho odoslanie naspäť do aplikácie. Prejdime teraz k definícii CSP.

Skripty sú celkovo tri, po jednom pre melódiu, pre dvojice tónov a pre akordy. Tieto procedúry definujú CSP pomocou pravidel z dynamickej časti. Vo všetkých troch sú použité obmedzenia tónu do zvolenej stupnice, vrátane pravidla pre priamu či nepriamu zhodu.

Skripty pre dvojice nôt a akordy sú prakticky rovnaké, líšia sa len v počte hľadaných nôt, a teda v počte aplikovaní pravidla harmonického intervalu. Pre dvojicu nôt ho stačí použiť raz, pri akorde až 3krát (pre každú dvojicu tónov raz). Pri určovaní melódie hľadáme melodický interval a k tomu potrebujeme poznať predošlú notu. Našťastie si je prehľadávač schopný zapamätať niekoľko minulých riešení a tak môžeme pravidlo melodického intervalu aplikovať. Kedže ale berieme do úvahy predošlú notu, musíme mať samostatný prehľadávač pre každý MIDI kanál. Ak by bol dostupný iba jeden melodický prehľadávač, mohli by nastať problémy.

Uvažujme situáciu, kde má MIDI kanál 1 hrať melódiu a kanál 2 akordový sprievod. Býva zvykom, že melódia je hraná minimálne o oktávu vyššie ako akordy. V momente, keď pre akord existuje len riešenie s jedným tónom, by sa použil rovnaký prehľadávač ako pre melódiu. Ako už vieme, melodický prehľadávač berie do úvahy predchádzajúci tón. Interval medzi tónom z akordu a tónom melódie môže presiahnuť oktávu a takéto melodické intervaly nie sú implementovaným pravidlom povolené. Vo výsledku by sme stratili notu z melódie alebo sprievodu napriek tomu, že riešenie mohlo existovať, ak by sa použil samostatný melodický prehľadávač pre každý kanál.

Do budúcna sa ráta so spoločným prehľadávačom pre melódiu, dvojice tónov aj akordy. Je totiž žiadúce, aby sa melodické intervaly aplikovali aj pre po sebe idúce súzvuky tónov. Tým by sa výsledný program zjednodušil a obsahoval by iba jednu procedúru s definíciou CSP a jeden univerzálny prehľadávač pre každý MIDI kanál.

5.3.4 Komunikácia so Strasheelou

Komunikácia medzi aplikáciou a programom, ktorý beží na OzServeri, prebieha pomocou protokolu Open Sound Cotrol. Tento protokol má schopnosť interagovať s inými systémami, je presný, flexibilný a určený pre komunikáciu medzi počítačmi, syntetizérmi a inými multimediálnymi

zariadeniami [25]. V tejto kapitole najprv priblížime špecifikáciu OSC a zadefinujeme formát posielaných správ. Potom sa pozrieme na ich spracovávanie v aplikácii a v Strasheele.

5.3.4.1 Špecifikácia OSC a formát posielaných správ

Technické informácie o OSC protokole uvedené v tejto kapitole sú voľne prevzané zo špecifikácie OSC verzie 1.0 a sú v plnom znení dostupné z [26]. OSC používa pre prenos informácii pakety. Paket pozostáva zo súvislého bloku binárnych dát a čísla, ktoré udáva veľkosť dát. Protokol je nezávislý na transportnej vrstve a spolieha sa na sieťový protokol (TCP alebo UDP), že dáta prenesie správne. Dáta v pakete musia byť typu OSC správa (message) alebo OSC balík (bundle). Typ paketu sa rozlišuje na základe prvého bajtu.

OSC správa sa skladá z adresového vzoru (OSC address pattern), typového reťazca (OSC type tag string) a argumentov (OSC arguments). Adresový vzor je reťazec, ktorý začína znakom '/'. Typový reťazec začína znakom ',' (čiarka), ktorý je nasledovaný sekvenciou symbolov predstavujúcich OSC dátové typy argumentov. Je vhodné spomenúť, ze niektoré staršie implementácie OSC vynechávajú typový reťazec. Argumenty sú reprezentované len ako sekvencia binárnych dát. Správa nemusí obsahovať argumenty.

Druhým typom paketu je OSC balík. Je zložený z reťazca „#bundle“, ktorý je nasledovaný časovým razítkom a ľubovoľným počtom elementov (OSC bundle elements). Element má určitú veľkosť a obsah, pričom obsahom može byť OSC správa alebo OSC balík. Je tak umožnené vytvárať hierarchickú štruktúru a preniesť ju po sieti ako jeden OSC paket.

Aplikácia využíva možnosti hierarchickej štruktúry a posielaný paket obsahuje balík so správami nôt v balíku so správou MIDI kanálu. Obsah paketu teda vyzerá nasledovne:

```
[
    /channel 2
    [
        /note 45 4 70
        /note 38 4 70
        /note 41 4 70
    ]
]
```

kde „[“ predstavuje začiatok a „]“ koniec OSC balíku. Pre názornosť sú pri balíkoch vynechané reťazce „#bundle“ a časové razítka. Pri OSC správach je vynechaný typový reťazec. Tieto informácie sú vo väčšine prípadov aj tak doplnené automaticky v jednotlivých implementáciach OSC. Správa MIDI kanálu ma adresový vzor „/channel“ a má jeden argument, číslo kanálu. Správy noty majú adresový vzor „/note“ a 3 číselné argumenty – výška, dĺžka a hlasitosť. Výška a hlasitosť korešpondujú s povoleným rozsahom pre MIDI noty (0-127) a dĺžka je udaná ako počet šestnástinových nôt (viď. kapitola 5.2.4).

5.3.4.2 Java a OSC

Pre prijímanie a odosielanie OSC paketov je v aplikácii použitá knižnica JavaOSC, ktorá je vytvorená skupinou Illposed Software. Zdrojové kódy knižnice sú súčasťou aplikácie, pretože boli pre moje účely trochu modifikované. Upravený bol zdrojový kód pre rozosielanie prijatých OSC paketov. Ten v prípade prijatia OSC balíku rekurzívne prehľadával elementy a rozosielal až jednotlivé OSC správy. V aplikácii však viem, aký vstup očakávam a spracovávam rovno prijatý balík. Zmeny sú v súlade s licenciou tohoto softvéru.

Po obdržaní správy rozhrania z generátoru sa OSC pakety posielajú iba v prípade, že sa nejedná o prázdnu správu a je spustený OzServer. Ak sú tieto podmienky splnené, je pre každý MIDI kanál v správe rozhrania vytvorený a odoslaný jeden paket. Dôvodom je odľahčenie programu bežiacom na OzServeri od logiky pre spracovanie viacerých balíkov MIDI kanálov. Prijatá správa rozhrania je dočasne uložená, aby mohla byť po obdržaní všetkých odpovedí pozmenená a preposlaná do zvukovej jednotky. Príprava paketu je za pomoci knižnice JavaOSC veľmi jednoduchá a intuitívna. Pre vytvorenie OSC správy stačí adresový vzor a argumenty. Balíky sa vytvárajú ako sada paketov (správ a balíkov). K odoslaniu pripraveného paketu je potrebné zavolať metódu `send()` objektu vytvorenému z triedy `OSCPortOut`.

O prijímanie sa stará objekt triedy `OSCPortIn`, ktorý pri prijatí paketu volá metódu pre jeho spracovanie. Keď zachytíme OSC paket odoslaný zo Strasheely, musíme aplikovať opačný postup ako pri jeho odosielaní. Udalosti rozhrania na príslušnom MIDI kanáli z uloženej správy rozhrania sú pozmenené hodnotami z balíku nôt. S odoslaním správy rozhrania do zvukovej jednotky sa čaká, kým neprídu OSC pakety pre všetky kanály. Počet prijatých paketov je vždy zhodný s počtom odoslaných paketov, pretože Strasheela odpovedá aj v prípade, že riešenie neexistovalo. Ak nastane takáto situácia, všetky pôvodné udalosti rozhrania pre daný MIDI kanál sú odstránené a vo výsledku sa vo zvukovej jednotke nič neprehrá.

5.3.4.3 Strasheela a OSC

Strasheela pre prácu s OSC používa dve UNIXové aplikácie: sendOSC a dumpOSC. To znamená, že hudobné jadro je závislé na systéme a celá aplikácia sa tak stáva platformovo závislou. Napriek tomu, že OSC protokol je úspešne implementovaný aj na platforme Windows, Strasheela momentálne nepodporuje žiadnu inú alternatívu.

Trieda `DumpOSC` implementuje Oz rozhranie k aplikácii dumpOSC, ktorá na stanovenom porte prijíma OSC pakety. Strasheela volá dumpOSC v terminále (xterm) a jeho výstup je pomocou aplikácie netcat posielaný načúvaciemu portu Oz kompilátoru. Toto zvláštne riešenie (pre ktoré mal však Torsten Anders dobrý dôvod) môže koncovým uživateľom mojej aplikácie znepríjemniť život. Pri sieťovom prenose výstupu dumpOSC do Oz kompilátoru sa implicitne ráta s IP adresami verzie 4. V mojom prípade však netcat za základné nastavenie považoval IPv6. Táto nezhoda spôsobila zastavenie prenosu dát z dumpOSC a celý proces tak zlyhal. Chyba sa naštastie dá opraviť

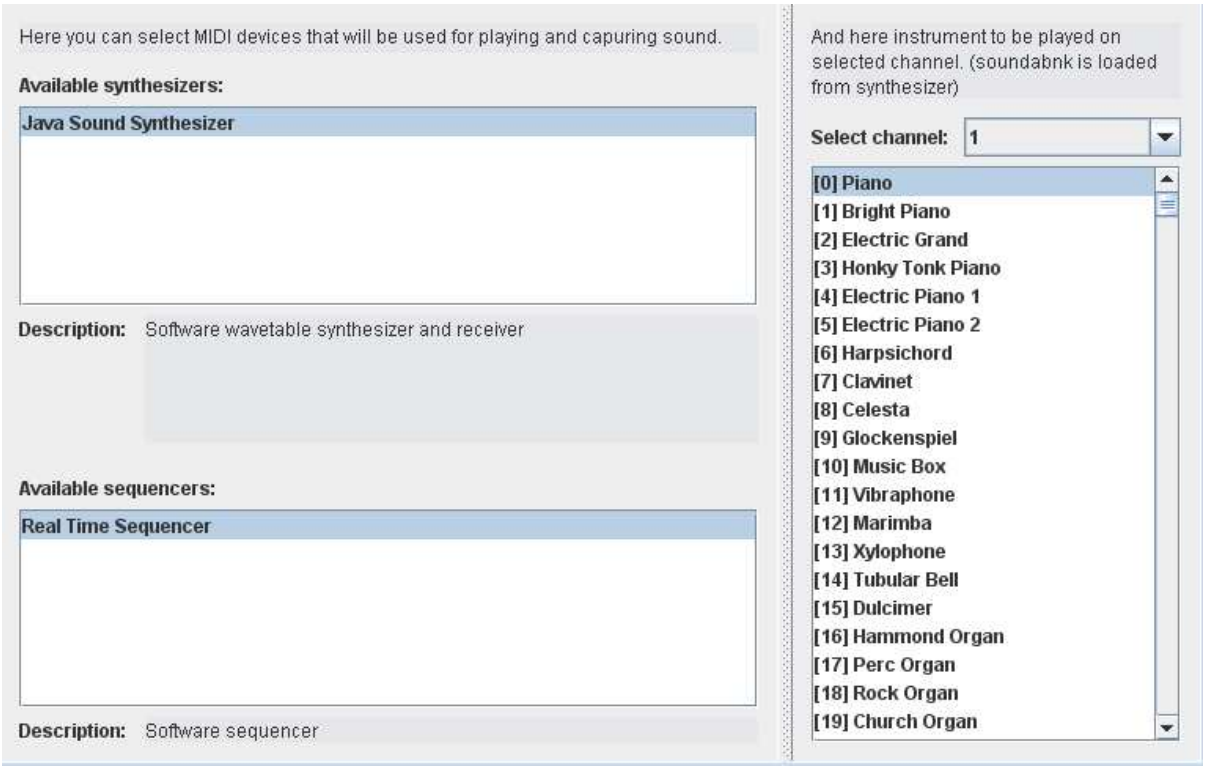
v zdrojovom kóde OSC funktoru OSC.oz pridaním parametru pri volaní aplikácie netcat. Textový výstup dumpOSC je interne prevedený na Oz hodnoty a programu je poskytnutý prístup k jednotlivým paketom. Podobne ako na strane aplikácie sa pri prijatí spustí procedúra, ktorá je popísaná v kapitole 5.3.3.3.

Strasheela poskytuje aj rozhranie `SendOSC`, ktoré sa stará o posielanie OSC paketov za pomoci aplikácie sendOSC. Program po zistení riešenia vytvorí paket s požadovaným formátom a jeho odoslanie je záležitosťou procedúry `send()` z triedy `SendOSC`. Momentálne sa z prenášaných údajov v Strasheele spracováva iba výška tónu, ostatné argumenty sú v správe noty zahrnuté pre prípad rozšírení hudobných pravidiel aj do oblasti dĺžky alebo hlasitosti tónov.

5.4 Zvuková jednotka

Úlohou zvukovej jednotky je zachytiť výstup hudobného jadra a umožniť jeho prehratie a zápis do súboru. Ide teda o posledný článok algoritmického generovania v tejto aplikácii.

Ovládací panel komponentu (obrázok 5.10) má časť pre výber MIDI zariadení a pre výber hudobných nástrojov. Dostupné MIDI zariadenia sú rozdelené do dvoch zoznamov, na syntetizéry a sekvencery. K zvolenému zariadeniu sa zobrazuje aj krátky popis. Výber hudobného nástroja je umožnený pre každý MIDI kanál. Zoznam nástrojov je načítaný zo syntetizéru a ak žiadny nie je dostupný, bude zoznam prázdny.



Obrázok 5.10 – Ovládací panel zvukovej jednotky

Pre prácu s MIDI zariadeniami sa používajú triedy z balíka `javax.sound.midi`. Ich pomocou je možné implementovať všetky potrebné operácie zvukovej jednotky. Syntetizér prijíma objekty `ShortMessage`, ktoré musia byť najprv nastavené metódou `setMessage()`. V prípade prehrávania tónu sú parametre tejto správy jej typ (`ShortMessage.NOTE_ON`), číslo MIDI kanálu, výška tónu a jeho hlasitosť. Tento objekt sa využije aj pri vytváraní objektu `MidiEvent` pre sekvencer. Okrem toho sa však predá aj informácia o čase, ktorý určuje, kedy udalosť nastala.

Existujú dva spôsoby ako zaznamenávať MIDI udalosti, ktoré sú úzko späté s formátom MIDI súboru. Udalosti sú zapisované do zvukových stôp (`Track`) a tie sú uložené v sekvencii (`Sequence`). MIDI súbor typu 0 má iba jednu stopu a v nej sú zaznamenané všetky udalosti. Typy 1 a 2 povoľujú v sekvencii aj viac stôp. Je tak možné rozdeliť udalosti do stôp podľa MIDI kanálu, na ktorom sa odohrali. Tento spôsob považujem za lepší a využívam ho v svojej aplikácii. Napr. v prípade rozšírenia zvukovej jednotky o editor nahraného hudobného materiálu by sme mohli pristupovať priamo k udalostiam jedného kanálu. V prípade, že nie je dostupný žiadny syntetizér, aplikácia nebude prehrávať zvuk. Ak nastane rovnaká sitúacia pre sekvencer, užívateľ nebude mať možnosť zapísať prehraný zvuk do súboru.

Zvuková jednotka sa musí starať aj o vytváranie udalostí uvoľnenia klávesy. Využíva sa rovnaký princíp ako v generátore pre blokovanie kanálu. Existuje teda určitá pamäť, do ktorej sa ukladajú všetky potrebné informácie k vygenerovaniu udalosti NOTE OFF. S každou prijatou správou rozhrania (ekvivalent úderu hodín v generátore) sa zmenší čakacia doba a po jej uplynutí sa vytvorená udalosť predá MIDI zariadeniam. Musíme si ale dať pozor na poradie správ na danom MIDI kanáli. Vo väčšine prípadov v jednom časovom okamihu nastane uvoľnenie aj stlačenie klávesy. Ak by dve po sebe idúce noty mali rovnakú hodnotu (výšku), mohlo by sa stať, že udalosťou NOTE OFF predošlej noty „vypneme“ nasledujúcu notu. Preto musia byť klávesy vždy uvoľnené pred nasledovnou udalosťou NOTE ON. Niekto by mohol namietať, že tento prístup nám pomôže iba v prípade okamžitého prehrávania syntetizérom. Veď sekvencer, a všeobecne formát MIDI, používa ako časový údaj číslo úderu (tick), ktorý je pre tieto udalosti rovnaký. Lenže žiadne MIDI zariadenie nedokáže presne v tom istom okamihu prehrať viac ako jednu udalosť a preto na ich poradí záleží. Časový rozdiel je ale tak malý, že to ľudské ucho nerozpozná.

6 Dosiahnuté výsledky

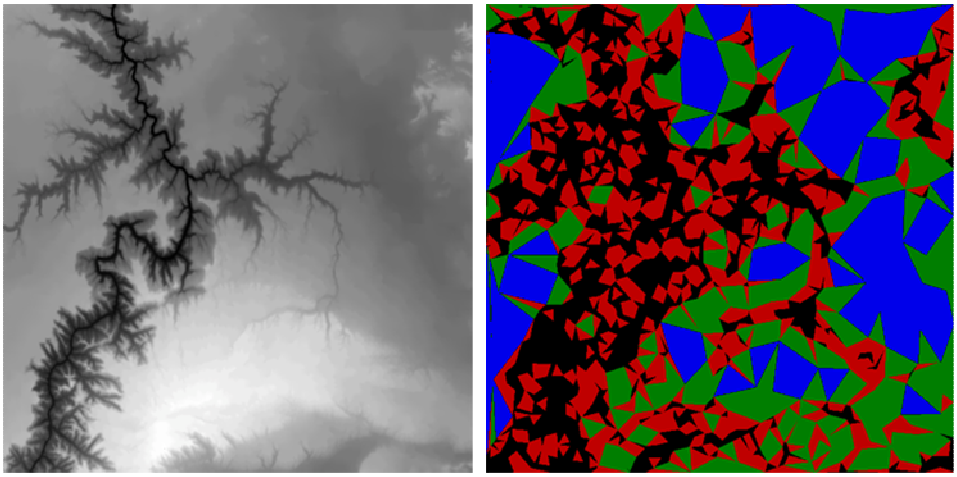
V tejto kapitole sa pokúsim zhrnúť, ako si implementovaná aplikácia poradila s úlohou algoritmickej kompozície melódie, akordového sprievodu a ich kombinácie, skladby. Táto neľahká úloha sa však nedá posúdiť objektívne, zhodnotenie je preto ponechané na samotnom čitateľovi a užívateľovi aplikácie.

Aby sme mali čo porovnávať, musíme najprv vygenerovať nejaké výstupy. Pre každú oblasť boli vytvorené dve dvojice nahrávok. Každá dvojica má rovnaké všetky vstupné parametre, až na jednu výnimku. Tou je nastavenie metódy určovania výšky tónu, pretože som zistil, že dokáže najviac ovplyvniť výstup aplikácie.

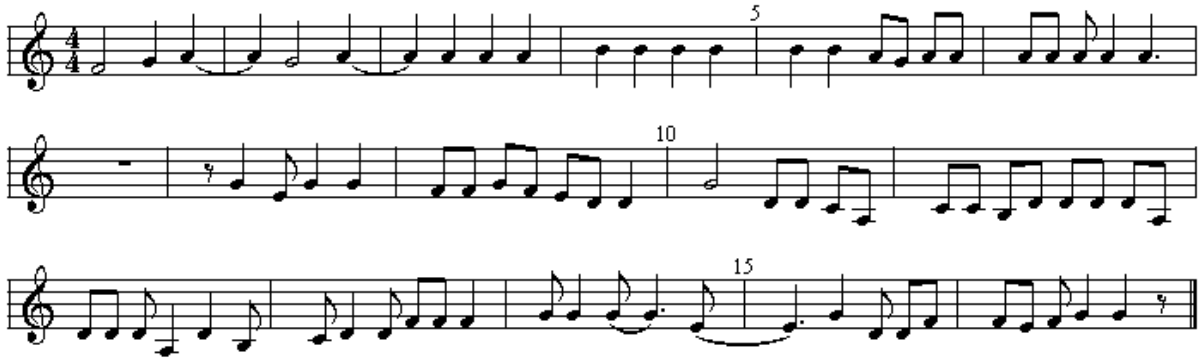
6.1 Melódia

Test 1

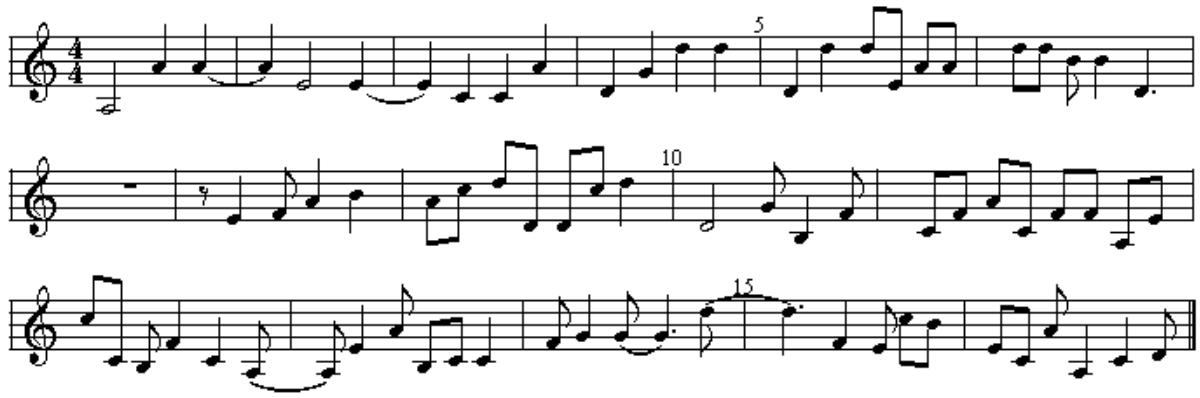
Agent má ponechané základné nastavenie. Rýchlosť jeho pochybu je 12 pixelov za štvrťovú notu a prehľadávací (explore) mód je vypnutý. V tomto teste je začiatočná pozícia agenta pravý dolný roh mapy a koncová pozícia je ľavý horný roh. Je použitý iba jeden MIDI kanál a ten má dve vrstvy. Obe použité mapy majú veľkosť 600x600 pixelov a sú zobrazené na obrázku 6.1. Výšková mapa bola vytvorená načítaním DEM súboru s dátami z oblasti Grand Canyon. Minimálna výška noty je 57 (A3) a maximálna hodnota je 75 (D#5). Mapa dĺžok nôt bola vytvorená pomocou vstavaného generátoru. Trojuholníková sieť bola vyfarbená štyrmi rôznymi farbami metódou zoradenia podľa veľkosti. Čierna farba predstavuje osminovú, červená štvrťovú a zelená pólovú notu. Modrá farba bola priradená pólovej pomĺčke. Melódia je v stupnici C dur a boli povolené všetky melodické intervaly.



Obrázok 6.1 – Mapy použité pri teste č.1



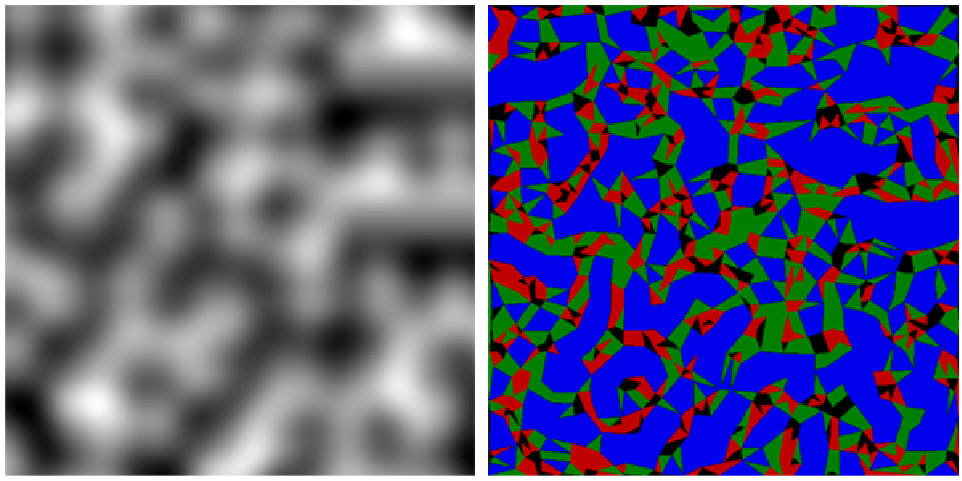
Notový zápis 1 – Test 1: Melódia s priamym mapovaním výšok tónov



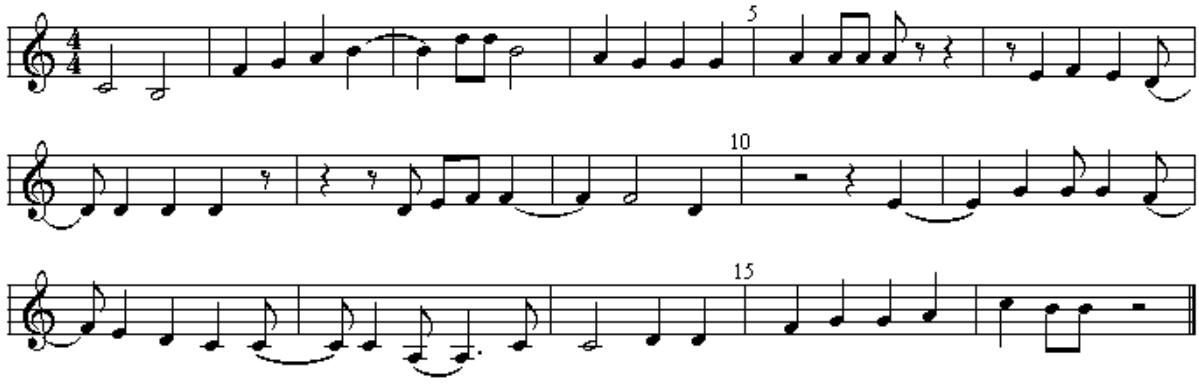
Notový zápis 2 – Test 1: Melódia s mapovaním výšok pomocou operácie modulo

Test 2

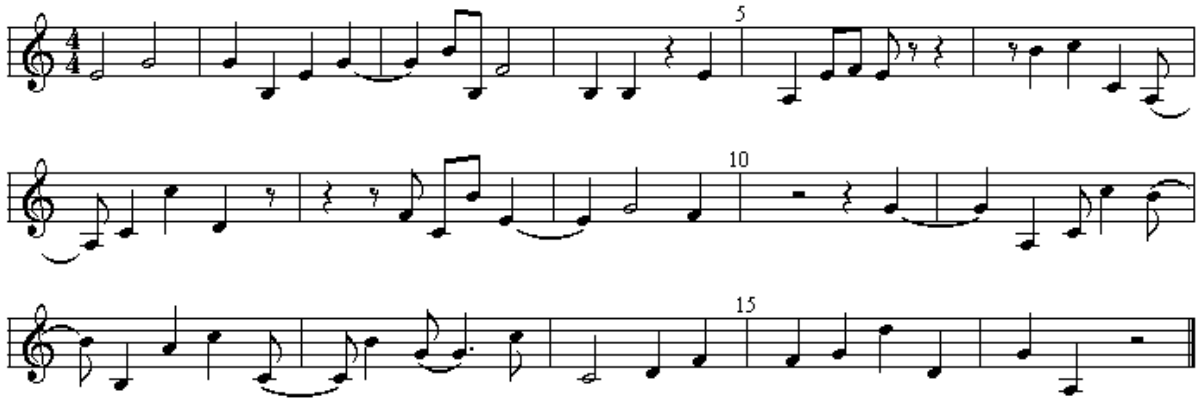
Začiatočná pozícia agenta je ľavý dolný roh mapy a koncová pozícia je pravý horný roh. Výšková mapa bola vytvorená generátorom výškových máp (frekvencia: 12, čas: 0). Mapa dĺžok nôt bola tiež vygenerovaná a vyfarbená metódou podľa veku trojuholníkov. Červená farba predstavuje osminovú, modrá štvrťovú a zelená pólovú notu. Čierna farba bola priradená pólovej pomĺčke. Ostatné nastavenia sú rovnaké ako v prvom teste.



Obrázok 6.2 - Mapy použité pri teste č.2



Notový zápis 3 – Test 2: Melódia s priamym mapovaním výšok tónov



Notový zápis 4 – Test 2: Melódia s mapovaním výšok pomocou operácie modulo

Z prvých dvoch testov je jasné, že spôsob určovania výšky tónu má na výstup naozaj veľký vplyv. Melódia vytvorená priamym mapovaním výšok kopíruje povrch terénu a pri jej počúvaní možme jednoducho spoznať, či terén klesal alebo stúpal. Nevýhodou ale je, že dostávame veľa rovnakých po sebe idúcich nôt. To je spôsobené malým rozsahom povolených hodnôt (18 poltónov) a problém sa teda dá jednoducho odstrániť zväčšením rozsahu.

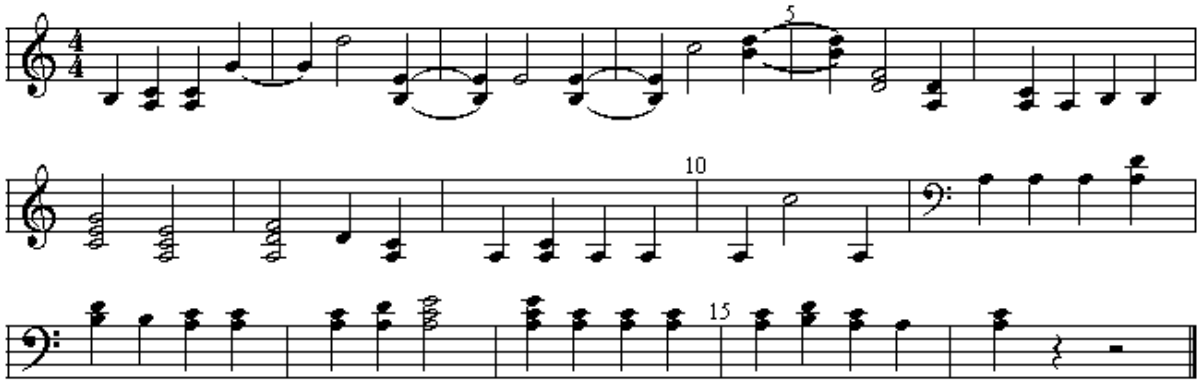
Zväčšenie intervalu nám však nemusí vždy vyhovovať, pekná melódia sa bezpochyby dá vytvoriť aj vrámci jednej oktávy. V tomto prípade sa oplatí použiť mapovanie výšok pomocou operácie modulo. Napriek tomu, že agent prešiel pri testoch po rovnakej trase, už zo skladby ani notového zápisu nedokážeme určiť vlastnosti terénu. Noty sú „rozhádzané“ v celom povolenom rozsahu a neopakujú sa tak často. V notovom zápise 4 sa ale v štvrtom takte objavila štvrťová pomlčka, ktorá nebola v tomto teste vygenerovaná pri priamom mapovaní. Pomlčku úmyslene vložilo hudobné jadro, pretože nebolo nájdené vhodné riešenie. Generátor musel vyprodukovať notu, ktorá je od predošlej noty vzdialená o viac ako oktávu (pripomínam, že nie sú povolené melodické intervaly väčšie ako oktáva).

6.2 Akordový sprievod

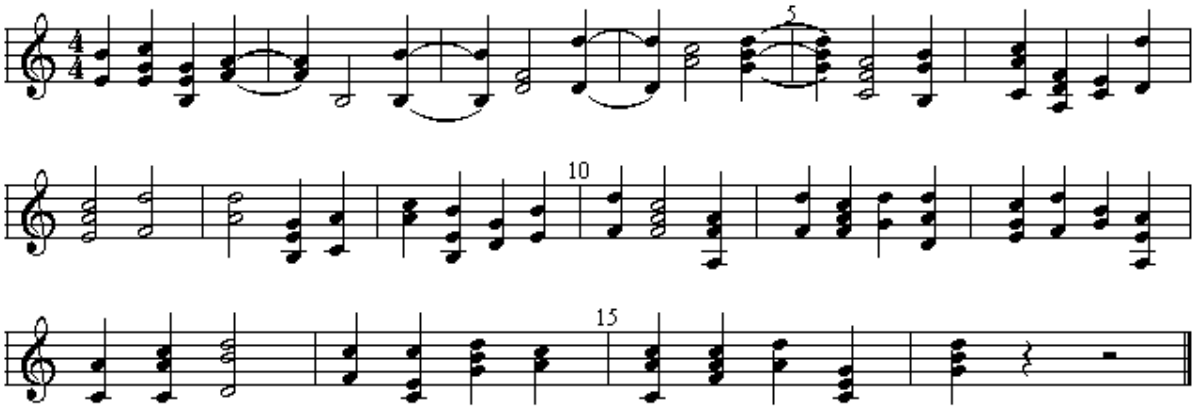
Test 3

Agent má ponechané základné nastavenie. Rýchlosť jeho pochybu je 12 pixelov za štvrťovú notu a explore mód je vypnutý. Začiatočná pozícia agenta je pravý horný roh mapy a postupuje sa po diagonále. Je použitý iba jeden MIDI kanál a ten má dve vrstvy. Obe použité mapy majú veľkosť 600x600 pixelov. Ako výšková mapa poslúžila fotografia záhradného prístrešku. Minimálna a maximálna výška noty sa oproti predošlím testom nezmenila. Mapa dĺžok nôt bola vytvorená aplikovaním algoritmu binárneho prahovania. Čierna farba predstavuje štvrťovú a biela pólovú notu. Akordový sprievod je tiež v stupnici C dur. Boli povolené harmonické intervaly malá a veľká tercia a sexta, kvarta, kvinta a oktáva.

Obrázok 6.3 – Mapy použité pri teste č.3



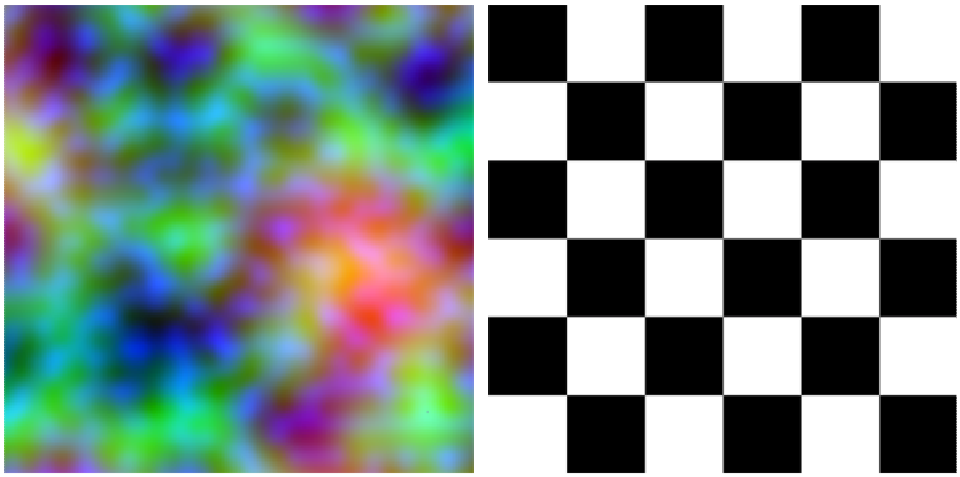
Notový zápis 5 – Test 3: Akordový sprievod s priamym mapovaním výšok tónov



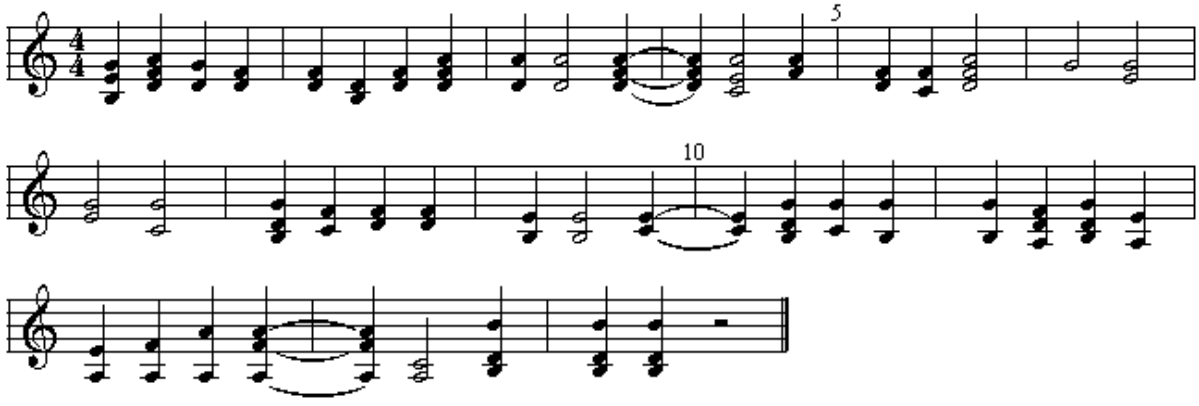
Notový zápis 6 – Test 3: Akordový sprievod s mapovaním výšok pomocou operácie modulo

Test 4

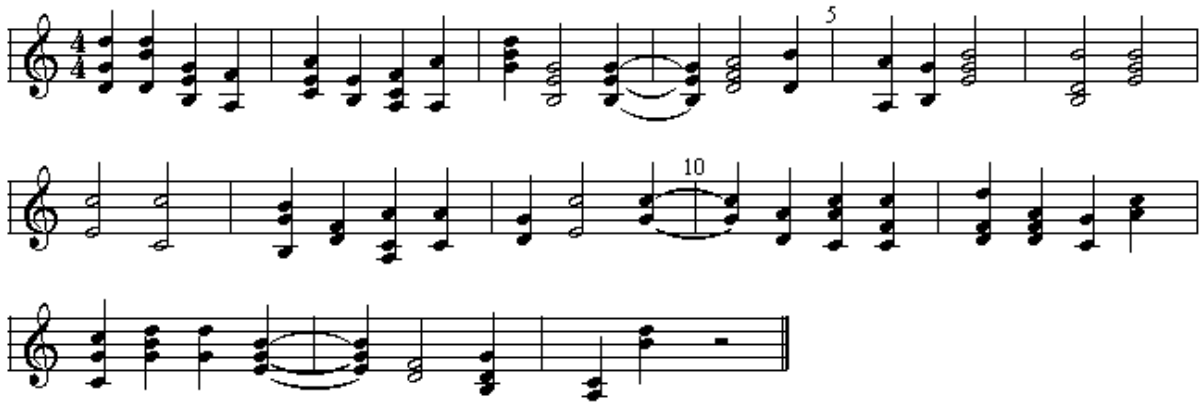
Výšková mapa bola vytvorená generátorom a pre každý farebný kanál boli zadané iné parametre (frekvencia, čas) – červený (307, 65), zelený (119, 9) modrý (42, 100). Mapa dĺžok nôt bola vytvorená v grafickom editore. Ostatné nastavenia sú rovnaké ako v teste č.3.



Obrázok 6.4 – Mapy použité pri teste č.4



Notový zápis 7 – Test 4: Akordový sprievod s priamym mapovaním výšok tónov



Notový zápis 8 – Test 4: Akordový sprievod s mapovaním výšok pomocou operácie modulo

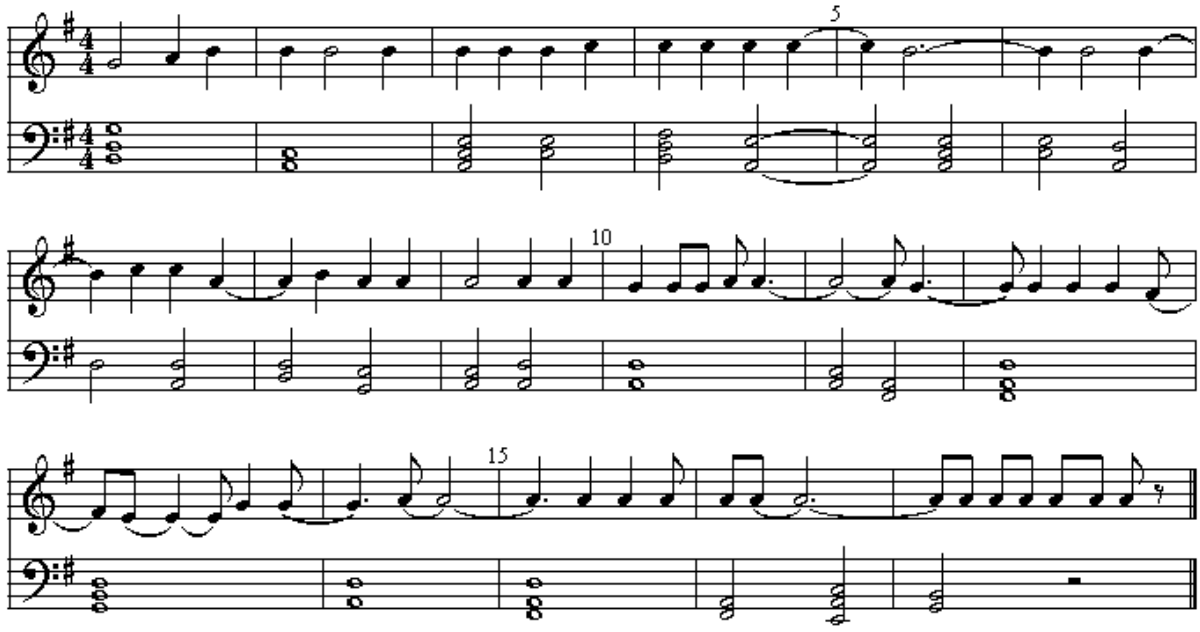
Tieto testy odhalili ďaľšiu slabú stránku metódy priameho mapovania. V treťom teste bolo nájdených len 5 riešení pre akordy, väčšina bola pre dvojice tónov (23) a v tesnom závese boli riešenia iba s jedným tónom (20). Situácia sa zlepšila v teste č. 4, kde bolo len jedno riešenie s jedným tónom. Výsledok je teda silne závislý aj na mape výšok. To však neberiem ako nevýhodu, pokladám to za korektné správanie. Pozitívnou správou ale je, že algoritmus pre postupné hľadanie riešení popísaný v kapitole 5.3.3 funguje spoľahlivo. Ďaľším poznatkom je, že pri použití priameho mapovania má väčšina po sebe idúcich akordov (alebo dvojíc tónov) aspoň jednu spoločnú notu. Podobne ako pri melódii to pripisujem malému rozsahu možných hodnôt tónov.

V prípade použitia operácie modulo sú výsledky veľmi zaujímavé a pestré. Len zriedka sa objaví riešenie s jedným tónom a akordy sa objavujú približne rovnako často ako dvojice nôt.

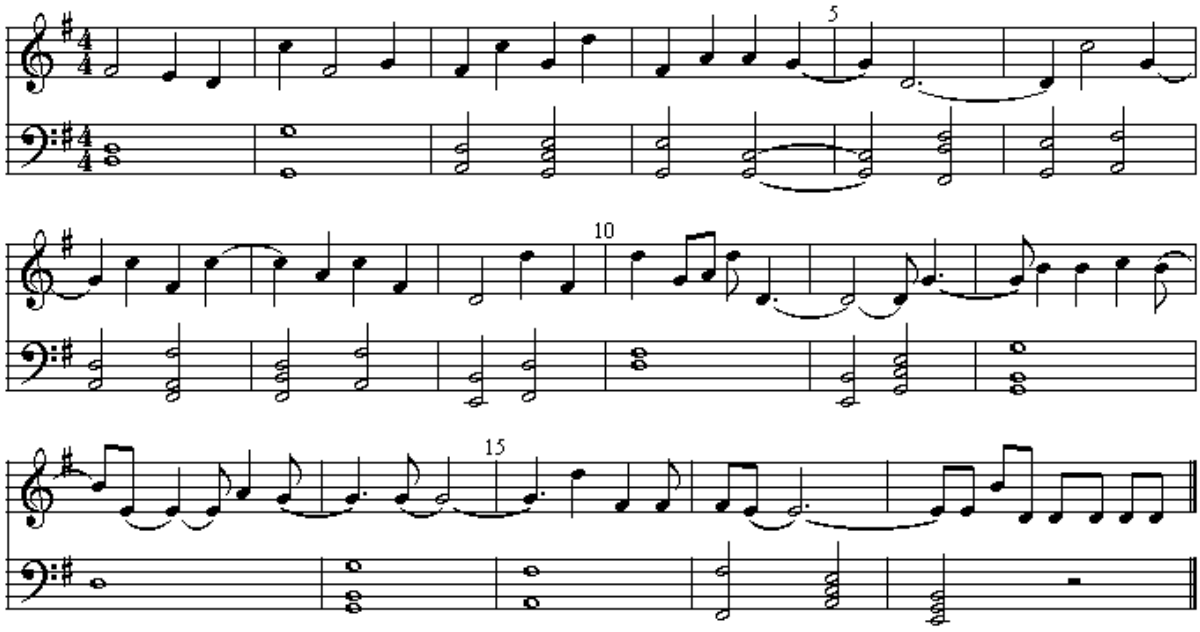
6.3 Skladba

Test 5

Agent má ponechané základné nastavenie. Rýchlosť jeho pochybu je 12 pixelov za štvrťovú notu a explore mód je vypnutý. Agent začína v pravom dolnom rohu mapy a koncová pozícia je ľavý horný roh. Sú použité dva MIDI kanály, jeden pre melódiu a druhý pre akordový sprievod. Na oboch kanáloch sú využité vrstvy výšok a dĺžok nôt. Mapy sú prevzaté z predošlých testov. Pre melódiu je použitá výšková mapa z prvého testu a mapa dĺžok z testu č. 2. Minimálna výška noty je 63 (D#4) a maximálna hodnota zostala na 75 (D#5). Červená farba predstavuje osminovú, modrá štvrťovú, zelená pólovú a čierna celú notu. Pre akordový sprievod som využil mapu výšok z testu č. 4 a dĺžky sú určené mapou z testu 3. Minimálna výška noty je 40 (E2) a maximálna je 56 (G#3). Biela farba bola priradená celej a čierna pólovej note. Skladba je v stupnici E mol, ktorá je zhodná zo stupnicou G dur. Boli povolené základné melodické a harmonické intervaly.



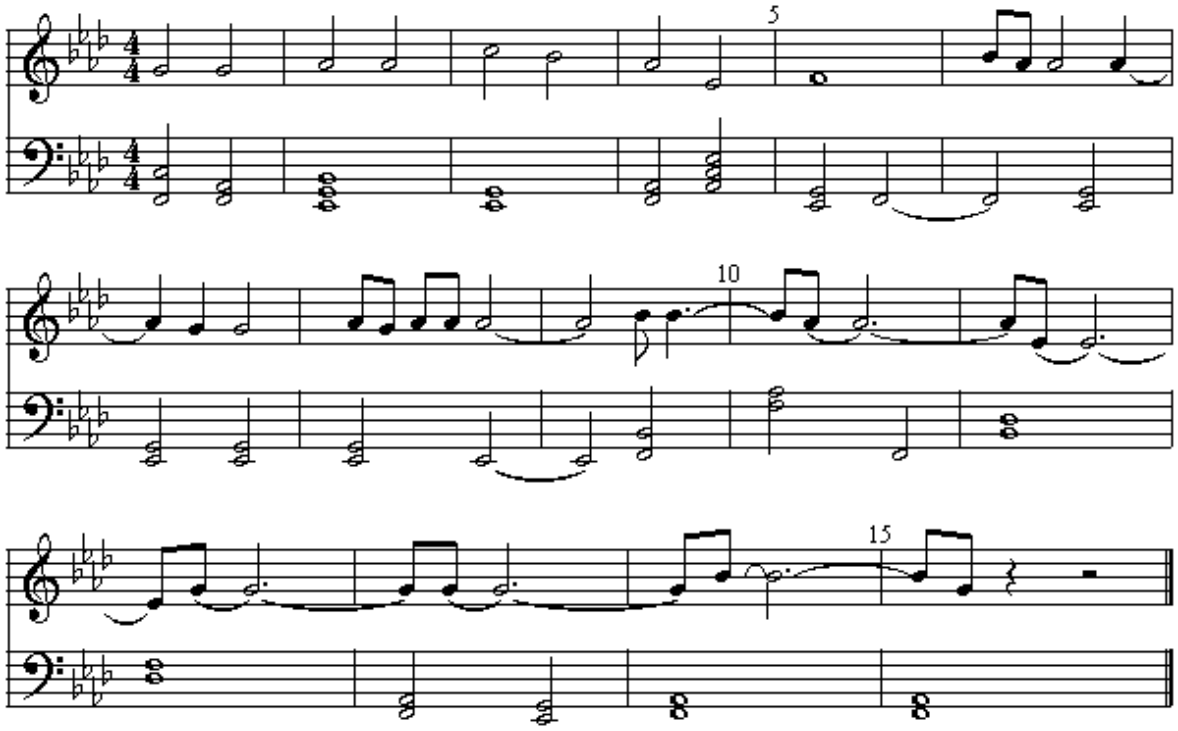
Notový zápis 9 – Test 5: Skladba s priamym mapovaním výšok tónov



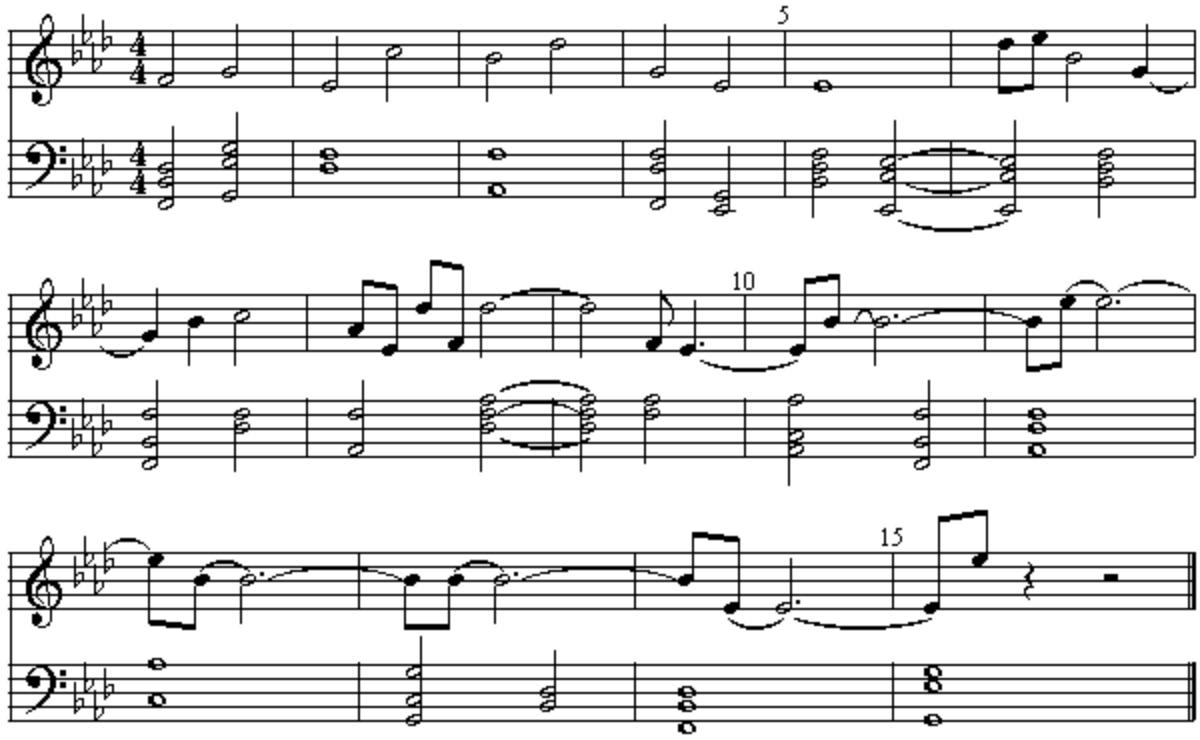
Notový zápis 10 – Test 5: Skladba s mapovaním výšok pomocou operácie modulo

Test 6

Záverečný test má zhodné nastavenia s testom 5. Boli použité mapy z testov pre melódiu a akordový sprievod, ktoré neboli súčasťou predchádzajúceho testu. Ich rozdelenie do máp pre melódiu a akordový sprievod súhlasí s použitím v testoch, z ktorých boli prevzaté. Skladba je v stupnici f mol, ktorá má 4 znížené tóny.



Notový zápis 11 – Test 6: Skladba s priamym mapovaním výšok tónov



Notový zápis 12 – Test 6: Skladba s mapovaním výšok pomocou operácie modulo

Súčasné prehrávanie (a nahrávanie) na dvoch MIDI kanáloch prebehlo bez akýchkoľvek problémov. Pri vhodnom nastavení všetkých parametrov je dokonca výsledná skladba príjemná na počúvanie. Najviac som sa obával disharmónii medzi melódiou a jej akordovým sprievodom, pretože

nie je implementované žiadne hudobné pravidlo, ktoré by obmedzovalo dianie na rôznych kanáloch. Tento úkaz našťastie nie je taký častý a pri dlhších nahrávkach bez problémov dokážeme nájsť pekné hudobné pasáže.

Pretože nemáme žiadne viackanálové pravidlo, nie je možné synchronizovať melódiu a jej sprievod. Môže teda nastať určitý rytmický posun, ako napr. v teste č.5 takt 10 a v teste č.6 takt 9. V oboch prípadoch bola vygenerovaná jedna osminová nota, ktorá spôsobila oneskorenie nástupu melódie v ďalšom takte. V niektorých skladbách to pôsobí zaujímavým a experimentálnym dojmom, v iných je tento jav rušivý. Našťastie sa aplikácia tohoto efektu dokáže po čase sama zbaviť, stačí vygenerovať správny počet nôt (v tomto prípade nepárny počet osminových nôt). Nepovažujem to však za vážny problém, úlohou aplikácie je poskytnúť čo najrozmanitejší hudobný materiál a záleží len na koncovom užívateľovi, čo z neho použije.

7 Záver

Práca poskytla prierez históriou formalizácie kompozície hudby a položila tak základ k štúdiu súčasných prístupov k jej algoritmickej tvorbe. Ku každému prístupu bol najprv uvedený stručný teoretický základ, ktorý prispel k pochopeniu príkladov z oblasti generovania hudby. Detailne bola rozpísaná technika programovania pomocou obmedzujúcich podmienok, ktorá bola využitá pri implementácii navrhnutého systému.

Mnou navrhovaná aplikácia bola dôkladne zanalyzovaná a rozdelená na niekoľko častí, ktoré boli následne úspešne implementované. Mojim cieľom bolo poskytnúť čo najväčšiu voľnosť pri používaní vytvorenej aplikácie. Výberom potrebných implementačných nástrojov a softvéru som, žiaľ, obmedzil koncovú platformu aplikácie na UNIXové systémy. Výhodou zostáva, že základné komponenty sú zapojené do rozhrania a pracujú samostatne. Systém tak podporuje nové implementácie generátora, hudobného jadra a zvukovej jednotky, ktoré môžu jednoducho nahradiť pôvodné komponenty a tak napr. odstrániť aj platformovú závislosť hudobného jadra. Do budúcna by som chcel aplikáciu pretvoriť do formy JAVA appletu a sprístupniť ju tak širšej verejnosti prostredníctvom priameho prístupu z internetu.

Mojou snahou bolo generovať hudbu bez nutnosti predošlého hudobného materiálu. Analýzou už vytvorených hudobních diel totiž dokážeme vytvárať len diela podobného žánru. Aplikácia má prvotne slúžiť ako experimentálny nástroj pre skladateľov, ktorý chcú nájsť nové a často nekonvenčné hudobné motívy. Umožňuje im preskúmať značnú časť hudobného priestoru a potenciál vygenerovanej hudby záleží len na vstupných dátach a ich úsudku. Implementácia takto rozsiahleho systému bola veľkou výzvou. Podarilo sa mi splniť všetky body zadania a s výsledkom práce som spokojný.

Systém, ktorý používa terénne a obrazové dáta ku generovaniu hudby, by mohol byť použitý napr. aj v herných aplikáciách s rozsiahlym svetom a vo všeobecnosti v ľubovoľných hrách s hudobným sprievodom. Ako ďalšia možnosť sa javí zkomponovať „piesene“ rôznych oblastí alebo dokonca štátov. Proces by sa po malých úpravách dal aj obrátiť, z hudby by sa tak dala generovať zaujímavá krajina.

Literatúra

[1] EPPERSON, Gordon. Encyclopaedia Britannica [online]. 2011 [cit. 2011-01-11]. Music. Dostupné z WWW: <http://www.britannica.com/EBchecked/topic/398918/music>.

[2] SIMONI, Mary. Algorithmic Composition : A Gentle Introduction to Music Composition Using Common LISP and Common Music [online]. Michigan : MI: Scholarly Publishing Office, 2003 [cit. 2011-01-11]. The History and Philosophy of Algorithmic Composition. Dostupné z WWW: <http://hdl.handle.net/2027/spo.bbv9810.0001.001>.

[3] VIČAR, Jan; DYKAST, Roman. Hudební estetika. Praha : Nakladatelství Akademie múzických umění, 2001. Antická hudební estetika, s. 79-98.

[4] VRKOČOVÁ, Ludmila. Slovníček základních hudebních pojmů : [názvosloví, formy, nástroje, instituce, dějiny, styly]. 2. vyd. [Česká republika] : Ludmila Vrkočová, 1995. 221 s ISBN 8090161111.

[5] SMITH, Timothy A. Sojurn [online]. 1996 [cit. 2011-01-11]. Anatomy of a Canon. Dostupné z WWW: <http://janus.ucc.nau.edu/tas3/canonanatomy.html>.

[6] APEL, Willi. Harvard Dictionary of Music. USA : Belknap Press, 1944. 958 s. Dostupné z WWW: <http://www.hup.harvard.edu/catalog.php?isbn=9780674375017.

[7] FILDES, Jonathan. BBC News [online]. 2008 [cit. 2011-01-11]. 'Oldest' computer music unveiled. Dostupné z WWW: <http://news.bbc.co.uk/2/hi/technology/7458479.stm>.

[8] DUNN, John . Algorithmic Art [online]. 1992 [cit. 2011-01-11]. Algorithmic Music . Dostupné z WWW: <http:algoart.com/music.htm>.

[9] VAN RANSBEECK, Samuel ; GUEDES, PHD, Carlos. Stockwatch:A Tool for Composition with Complex Data. Parsons Journal for Information Mapping [online]. 2009, 3, [cit. 2011-01-11]. Dostupný z WWW: <http://piim.newschool.edu/journal/issues/2009/03/pdfs/ParsonsJournalForInformationMapping_VanRansbeeck-Samuel+Guedes-Carlos.pdf>.

[10] JÄRVELÄINEN, Hanna. Algorithmic Musical Composition [online]. Helsinki, 2000. 12 s. Oborová práce. Helsinki University of Technology. Dostupné z WWW: <http://www.tml.tkk.fi/Studies/Tik-111.080/2000/papers/hanna/alco.pdf>.

[11] Algorithmic Composer [online]. Květen 2010 [cit. 2011-01-11]. Algorithmic Composition: Markov Chains in Max MSP. Dostupné z WWW: <http://algorithmiccomposer.blogspot.com/2010/05/algorithmic-composition-markov-chains.html>.

[12] GRINSTEAD, Charles M.; SNELL, J.Laurie. Introduction to Probability [online]. USA : The American Mathematical Society, 1997 [cit. 2011-01-11]. Dostupné z WWW: <http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/amsbook.mac.pdf>.

[13] MAURER IV., John A. Stanford Edu. [online]. 1999 [cit. 2011-01-11]. The Influence of Chaos on Computer-Generated Music. Dostupné z WWW: <https://ccrma.stanford.edu/~blackrse/chaos.html>.

[14] CLARK, James . Advanced Programming Techniques for Modular Synthetizers [online]. 2003 [cit. 2011-01-11]. Algorithmic Composition. Dostupné z WWW: <http://www.cim.mcgill.ca/~clark/nordmodularbook/nm_algorithmic.html>.

[15] NIERHAUS, Gerhard. Algorithmic Composition : Paradigms of Automated Music Generation [online]. Německo : Springer-Verlang/Wien, 2009 [cit. 2011-01-11]. Dostupné z WWW: <http://www.scribd.com/doc/27068203/Algorithmic-Composition>.

[16] Oktopus [online]. 2006 [cit. 2011-01-11]. Dostupné z WWW: <http://www.oktopus.hu/imgs/MANAGED/Hangtechnikai_tudastar/The_MIDI_Specification.pdf>.

[17] CHADABE, Jeffrey Prof. Introduction to Computer Music: Volume One [online]. 2010 [cit. 2011-01-11]. Chapter Three: MIDI. Dostupné z WWW: <http://www.indiana.edu/~emusic/etext/MIDI/chapter3_MIDI.shtml>.

[18] KELLY, John J. Essence of Logic [online]. London,New York : Prentice Hall, 1997 [cit. 2011-01-11]. Dostupné z WWW: <http://thebookshq.com/books/essence-logic-kelly.html>.

[19] Constraint Satisfaction Problems [online]. [cit. 2011-01-11]. Dostupné z WWW: <http://aima.cs.berkeley.edu/newchap05.pdf>.

[20] ANDERS, Torsten . Composing Music by Composing Rules : Design and Usage of a Generic Music Constraint System [online]. Belfast : Queen´s University Belfast, 2007. 244 s. Diplomová práce. School of Music and Sonic Arts. Dostupné z WWW: <http://strasheela.sourceforge.net/documents/TorstenAnders-PhDThesis.pdf>.

[21] ANDERS, Torsten. Strasheela [online]. 2009 [cit. 2011-01-11]. Dostupné z WWW: <http://strasheela.sourceforge.net/strasheela/doc/index.html>.

[22] WRIGHT, Matt. Open Sound Control [online]. 2002 [cit. 2011-01-11]. Dostupné z WWW: <http://opensoundcontrol.org/>.

[23] PERLIN, Ken. Improving Noise. ACM SIGGRAPH [online]. 2002, Volume 21 Issue 3, [cit. 2011-05-14]. Dostupný z WWW: <http://mrl.nyu.edu/~perlin/paper445.pdf>.

[24] SHOJAEE, D.; HELALI, H.; ALESHEIKH, AA. Triangulation for Surface Modelling. Ninth International Symposium on the 3D Analysis of Human Movement [online]. 2006, [cit. 2011-05-15]. Dostupný z WWW: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.2934&rep=rep1&type=pdf>.

[25] Opensoundcontrol.org [online]. 2002 [cit. 2011-05-16]. Introduction to OSC. Dostupné z WWW: <http://opensoundcontrol.org/introduction-osc>.

[26] Opensoundcontrol.org [online]. 2002 [cit. 2011-05-16]. The Open Sound Control 1.0 Specification. Dostupné z WWW: <http://opensoundcontrol.org/spec-1_0>.

Zoznam príloh

Príloha 1. CD so zdrojovými súbormi, návodom k inštalácii a ukážkami hudobných výstupov.