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

ALGORITMICKÉ GENEROVÁNÍ ESTETICKÝCH RYTMICKÝCH SEKVENCÍ

GENERATING AESTHETICAL RHYTHMICAL SEQUENCES USING ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELORS'S THESIS

AUTOR PRÁCE
AUTHOR
JAKUB MAHNERT

VEDOUCÍ PRÁCE
SUPERVISOR
ING. VÍTĚZSLAV BERAN PHD.

BRNO 2016

Abstrakt

Cílem projektu je navrhnout a implementovat systém, který se snaží potvrdit, nebo vyloučit domněnku, že je možné generovat unikátní rytmické sekvence, které člověk vnímá jako estetické. V práci je potom navržen a implementován framework, umožňující tvorbu algoritmů, generujících rytmické sekvence technikou kompozice. V jeho rámci jsou potom navrženy a implementovány algoritmy, zkoumající v uživatelském testování korelaci pravidelnosti formy a vlivů náhody na výslednou míru estetičnosti sekvence.

Abstract

The goal of this project is to design and implement a system, suited for proving or disproving the claim that it is possible to generate unique rhythmical sequences, which a human will perceive as aesthetical. In the project, a framework, aiding the design of such algorithms that create rhythmical sequences using composition was implemented. On top of it, several such algorithms were implemented with the notion to test their performance using a custom testing front end solution. From the gained data, the project tries to find correlations between the aspects of the algorithms with the resulting perceived aesthetical value.

Klíčová slova

algoritmus, estetika, rytmus, sekvence, webová aplikace, Ruby, funkcionální programování

Keywords

algorithm, aesthetic, rhythm, sequence, web application, Ruby, functional programming

Citace

Mahnert Jakub: Algoritmické generování estetických rytmických sekvencí, bakalářská práce, Brno, FIT VUT v Brně, 2016

Algoritmické generování estetických rytmických sekvencí

Prohlášení

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

v Brně dne ……

……
Jakub Mahnert

Poděkování

Rád bych poděkoval zejména Anastasii za podporu, panu Schmebulockovi za trpělivost, svému vedoucímu za fundované podněty, Marii a Janě.

Obsah

Úvod

Cílem projektu je prozkoumat varianty řešení, zamyslet se nad algoritmy, analyzovat proces a následně navrhnout a implementovat systém, který se snaží potvrdit, nebo vyloučit domněnku, že je možné generovat unikátní rytmické sekvence, které člověk vnímá jako estetické.

I přes to, že se vybízí řešení využívající tzv. *knowledge-based* algoritmy (například rekurentní neuronová síť či jiné algoritmy založené na strojovém učení) byla tato varianta již v úvodní fázi vyloučena (více v kapitole **Specifika tohoto řešení**) - algoritmy a procesy, generující samotnou sekvenci jsou tedy analyzovatelné a kategorizovatelné pouze na základě úvahy, čehož bude v závěru práce využito k jejich evaluaci.

Vytvořený systém je potom paralelním modulárním sekvencérem, komponujícím rytmické stopy, které jsou následně vizualizačním modulem zobrazeny v podobě animace.

V rámci důkazu hlavního cíle projektu bude v projektu paralelně implemenováno šest variant výpočetních jader sekvencéru jako zvolený výběr z možných přístupů k problematice. Tyto jádra, respektive výsledky jejich činnosti jsou pak v rámci uživatelského testu porovnány a ohodnoceny a jeho výsledek je poté zhodnocen v kontextu vlastností jednotlivých jader.

Obsahem této práce je potom nejprve vymezení dalších cílů projektu a záměrů, vedoucích k jejich dosažení; dalších specifik, omezujících či specializujících směr řešení a také vysvětlení mezioborových vztahů. Práce se následně zaobírá návrhem systému a jeho nutných komponent nejen z hlediska dosažení cílů, ale i z hlediska jeho formálních atributů a vlastností; hovoří také o technologiích, knihovnách a postupech, které byly k implementaci zvoleny, tyto rozebírá a hodnotí jejich vhodnost pro aplikaci navržených systémů a algoritmů a dosažení cíle. V neposlední řadě pak práce zkoumá i samotnou implementaci aplikace, zaměřuje se na specifika řešení z pohledu softwarového inženýrství a představuje jeho silné i slabé stránky. Závěr

práce je věnován zejména detailnímu popisu způsobu ověření domněnky, zamýšlí se nad získanými daty z uživatelského testování a jejich signifikancí.

Generování estetických vzorů

Vzhedem k více možným validním způsobům evaluace míry estetičnosti objektu je vhodné hned v úvodu práce popsat, rozeznat a definovat pojem estetika, a to v kontextu bakalářské práce studenta technického oboru.

Díky transformující se definici oboru estetiky jako pojmu ovlivněném současnou úrovní chápání a porozumění není možné přímo a jednoznačně rozeznat a vydělit z množiny celku podmnožiny estetických či neestetických prvků - viz citace:

“Je zřejmé, že estetika je vědou o estetické funkci a estetickém postoji, ale v žádném případě ji nemůžeme pojímat jako vědu, která by se striktně zaobírala tvrzením, co je krásné a co již do této kategorie nepatří.“ **[2]**

V rámci kontextu této práce bude tedy estetičnost vjemu posuzována arbitrárním lidským testem, který ovšem nepracuje s pojmem estetika tak, jak jej definují a popisují odborná díla **[3]**, ale spíše slouží k porovnání jednotlivých výsledků a jejich líbivostí subjektivním lidským okem (viz kapitola **Testování**).

Existující systémy

Z identifikovaných a prozkoumaných informací vyplývá, že většina stávajících systémů a projektů pohlíží na pohled z opačného úhlu, než z jakého vnímá estetičnost díla touto prací. V posledních letech vzrůstá počet vědeckých prací, zabývajících se tematikou estetična a vnímání téhož lidským mozkem - zejména ale na principu rozeznávání stávajících estetických děl **[4] [5]** a hodnocení jejich míry estetičnosti (resp. jejím porovnávání). Tyto jsou ale zpravidla založeny na *knowledge-based* systémech - tedy typicky tzv. *deep* (hlubokých) neuronových sítích (v případě analýzy obrazu zejména konvolučních), které však vzhledem k vnitřnímu

fungování umělých neuronových sítí a jejich zaměření na arbitrární, lidským úsudkem obtížně analyzovatelné vzory pracují s pojmem estetika spíše nepřímo.

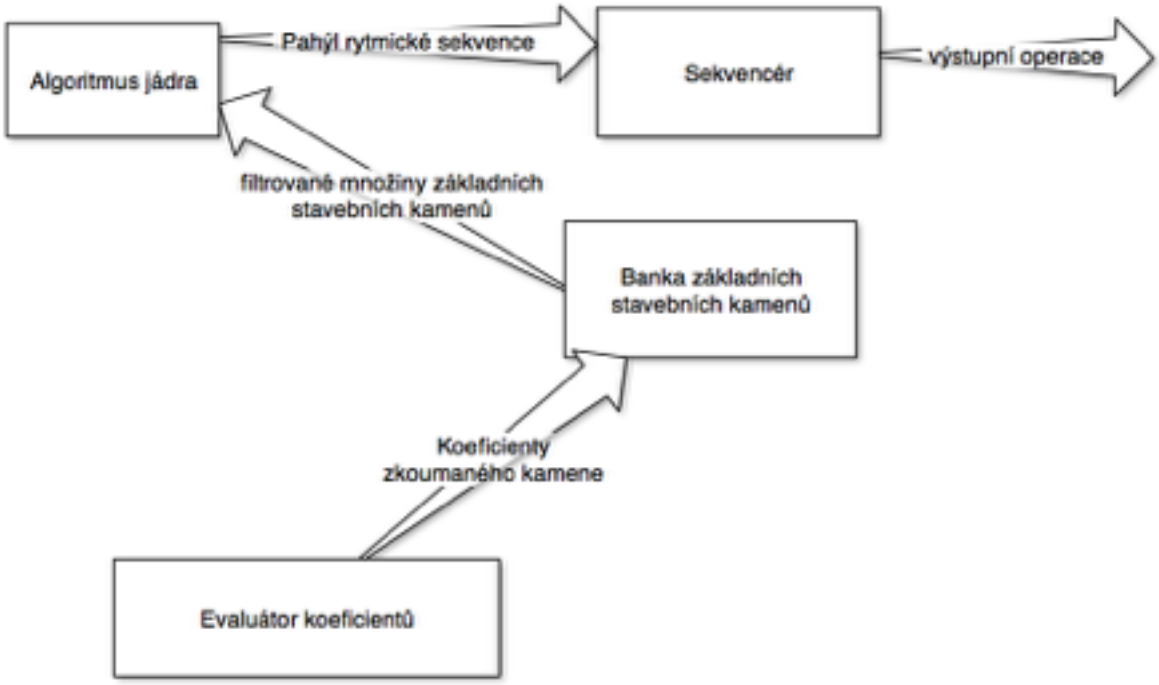
Existují také řešení, zabývající se generováním obecných sekvencí na základě analýzy vzorové banky sekvencí **[7]**, ale vzhledem k restrikcím, daným zadáním jde zcela zřejmě o úplně jinou disciplínu.

Je možné naleznout i řešení, zabývající se generováním specifických podskupin rytmických sekvencí - například rytmů, užívaných zejména v latinskoamerické či africké hudbě **[8],** které mají základy v Euklidově algoritmu. Tato řešení projekt neprozkoumává, zejména kvůli zvolené formě abstrakce - více viz kapitola **Jádro**.

Na trhu je také řada proprietárních řešení, zabývajících se generováním, rozeznáváním nebo klasifikováním rytmických sekvencí - počínaje jednoduchými algoritmy **[9]** až po komplexní automatické arpegiátory **[10].** Algoritmy, generující sekvenci jsou však v rámci *closed source* licence nezveřejněny, a proto nejsou v rámci této práce zkoumány a porovnávány s navrženým řešením.

Návrh řešení a architektury systému

Vyjma samotné implementace algoritmů kreativního procesu bylo nutné navrhnout a vytvořit rámcový systém - *framework*, který umožní abstrakci algoritmů a jejich oddělení od nutných režijních operací.



obr. 1 - architektura systému v zjednodušeném blokovém schématu

Specifika tohoto řešení

Na rozdíl od některých zmiňovaných projektů - které mají za úkol rozeznat estetickou hodnotu obrázku na základě identifikace vnitřních struktur neuronovou sítí - je cíl této práce opačný - sestavit systém, který je schopný konvenčním analyzovatelným algoritmem takový vjem připravit - a následně tento systém ohodnotit, rozeznat a popsat jeho typické znaky.

Proces tvorby tohoto systému byl ve fázi jeho analýzy rozdělen na několik dílčích kroků:

1. Analýza lidského kreativního procesu
2. Rozeznání řemeslných a invenčních částí procesu
3. Návrh a implementace nosného systému
4. Simplifikace řemeslné části procesu a její specifikace v podobě algoritmu
5. Implementace algoritmu a vizualizace jeho výsledků
6. Iterativní testování a optimalizace algoritmu

Rozdělení a rozeznání řemeslných od invenčních částí procesu probíhá na základě definice kreativního procesu: *"..to create consists of making new combinations of associative elements which are useful."* **[6]**, jakožto i jeho formální rozdělení do třech hlavních procesů - proces náhody, proces podobnosti, proces kombinace a proces mediace **[6]**. Z těchto třech procesů se pak tato práce zaměřuje zejména na procesy podobnosti a kombinace - tedy proces velmi blízký procesu klasicky řemeslnému, kdy je výtvor sestavován jako homonymní systém - subjektivní kopie - vzoru **[6]**.

V zájmu co největšího přiblížení k tomuto typu lidské tvorby byl systém modelován tak, aby algoritmus, generující estetický vjem pracoval s netriviálními základními stavebními prvky (v případě našeho systému členy množiny banky rytmů), které ovšem zvládá rozeznat a ohodnotit (ačkoliv nerozumí jejich vnitřní struktuře). Takový systém potom vytváří estetický výrobek na základě kombinování a skládání těchto základních kamenů - a právě snaha identifikovat a popsat takový algoritmus, který vytváří unikátní netriviální estetické vjemy formou modelování řemeslné části kreativního procesu bez využití knowledge based systému je cílem této práce.

Konfigurace aplikace

Pro běh programu je třeba definovat několik konstant, užitých během výpočtů. Jedná se zejména o konfigurace tzv. *nástrojů,* kde každý nástroj reprezentuje zjednodušený obraz kusu studiového osvětlení.

Takový konfigurační soubor potom popisuje jeho vlastnosti, ty má pak k dispozici příslušný algoritmus *jádra* (viz níže) a s jejichž pomocí filtruje základní stavební kameny na vhodnou podmnožinu.

Cílem navržené konfigurační techniky je pak rozlišení jednotlivých *nástrojů*, a to pak zejména v kontextu množiny těch, které byly v rámci práce implementovány.

```
def strobe
    {
        'on' => {
            'coef' => 0.3,
            'operator' => '=',
        },
        'front' => {
            'coef' => 0.75,
            'operator' => '=',
        },
        'heating' => nil,
        'pause' => {
            'coef' => 0.6,
            'operator' => '>'
        },
    }
end
```

obr. 2 - příklad konfiguračního objektu nástroje Strobe

Viz tento příklad konfigurace, zobrazující nutné položky, které musí konfigurace obsahovat - klíče, odpovídající patřičným vlastnostem nástroje; koeficienty, odpovídající příslušné hodnotě na stupnici daného atributu; a operátory, reprezentující výraz které jsou zapotřebí k navrženému robustnímu systému filtrování (viz kapitola **Banka rytmů**).

Je zřejmé, že navržené konfigurační řešení připomíná zejména velmi jednoduchý doménově specifický jazyk, jehož implementace je užito v rámci filtrování rytmů. Kvůli formálním nedostatkům implementace však v rámci této práce hovoříme spíše o soustavě konfigurace.

Banka rytmů

Základním stavebním kamenem systému je *Banka rytmů*, která má za úkol zastřešovat komunikaci s repozitářem rytmů a evaluaci těchto rytmů - základních stavebních kamenů - na základě evaluačních metod.

Řešení variantou kompozice ze základních stavebních kamenů bylo zvoleno na základě jednoduché úvahy. Jedná se o omezení možnosti algoritmu vnést do sekvence nějakou z elementárních chyb kompozice rytmu - tedy jevu, který by lidský mozek typicky jednoznačně podvědomě označil za neestetický.

Výpočetní jádro už potom komponuje výsledný rytmus pouze ze *zaručeně* estetických mikrosekvencí - základních stavebních kamenů. Tato technika však s sebou nese i své limitace - z výsledné množiny vygenerovatelných sekvencí pravděpodobně zmizí velká část takových, které by většina typicky označila za “velmi neestetické”, což na první pohled přináší úskalí během testování a evaluace.

Evaluace rytmu

Pro zjednodušení rozhodování vypočetního jádra je třeba, aby byly základní rytmy ohodnoceny. Ohodnocením se myslí ocenění rytmu číselnou hodnotou v několika navrhnutých dimenzích:

1. Poměr úderů vůči pauzám - **on** koeficient
2. Poměr úderů v blízkosti hlavní, těžké doby vůči poměru úderů v blízkosti lehkých dob - **front** koeficient
3. Poměr nejdelší nepřetržité pauzové sekvence vůči délce sekvence - **pause** koeficient

Zamýšlený cíl této množiny je pak zejména volná klasifikace rytmů formou kernelové metody **[11]**. Ta přiřazuje každému členu množiny hodnotu v jiných, než původně zadaných dimenzích s cílem dosáhnout stavu, kdy je možné výsledný N-dimenzionální prostor P rozdělit podle pravidel nástroje D na další N-dimenzionální prostory K1 až KX za pomoci (N - 1)-dimenzionálních útvarů tak, aby každý

podprostor K obsahoval prvky se společnými rysy. Pro kreativní algoritmus, generující sekvenci podle pravidel nástroje D je pak výběr vhodného rytmu pouze otázkou výběru vhodného podprostoru K - a základní rytmus, splňující pravidla nástroje D je pak kterýkoliv z členů množiny obyvatel vhodného prostoru K, protože tyto základní rytmy jsou z pohledu pravidel nástroje D zaměnitelné.

Tato množina zcela očividně nepokrývá všechny pozorovatelné atributy rytmů a lze nalézt takovou dvojici rytmů, jejichž ohodnocení se ve všech dimenzích shoduje, ale které nejsou mozkem vnímány jako zaměnitelné při tvorbě sekvence. Během iterativní implementace, kdy byla počáteční množina základních kamenů postupně rozšiřována o další a složitější rytmy a množina koeficientů rozšiřována o další dimenze ohodnocení rytmů vyplynulo, že jde vzhledem k vlastnostem a velikosti hodnocené množiny o rozdělení dostatečné.

Vyjma koeficientů **on** a **pause**, popisujících empirické vlastnosti rytmů (poměr not vůči pomlkám a nejdelší sekvenci pomlk) je v množině hodnotících funkcí také funkce, počítající koeficient **front.** Tento arbitrární atribut je navržen na základě úvahy, popisující a analyzující základní rytmus moderní populární hudby - tedy úder na první a úder na třetí dobu, kdy však každý z nich přísluší jinému nástroji (první dobu hraje typicky basový nástroj či hluboký buben, třetí pak výškový nástroj či tzv. *vířivý buben*). Rytmické stopy těchto nástrojů jsou pak ostatními navrženými hodnotícími funkcemi vnímány obdobně - a až doplněním evaluace v podobě **front** koeficientu získáme úplné rozčlenění množiny základních vstupních rytmů do třech dimenzí v podobě výsledku kernelové metody.

Sekvencér

Modul sekvencéru je v rámci sestavovaného systému mikroframeworkem, skrz jehož rozhraní jsou v rámci implementace jádra obsluhovány ostatní části systému. Jako takový musí být zejména modulární (což umožní uzpůsobit každou implementaci jádra zamýšlenému cíli), robustní (aby zvládl uspokojit všechny požadavky případného uživatelského jádra na kompozici rytmů, jejich zplošťování, normalizaci temp a podobně) a v neposlední řadě testovatelný formou jednotkových

a integračních testů - zejména kvůli zvolené formě aplikace a agilní metodice vývoje, ale i v rámci zásad a praktik softwarového inženýrství.

Modul sekvencéru také musí vystavit rozhraní, popisující vstupně - výstupní operace systému, případně vystavit rozhraní, komunikující s modulem, který tyto zastřešuje.

Je vhodné modul sekvencéru navrhnout s důrazem na jeho bezstavovost - tedy tak, aby byla každá operace, k jejímuž vykonání se vystavením rozhraním zaváže reprezentována kompozicí matematických funkcí. Taková bezstavová implementace v podobě funkcionální pipeline pak přináší nejen výhody z úhlu pohledu disciplíny softwarového inženýrství (mimo jiné lepší testovatelnost a rozšiřitelnost), ale i z pohledu optimalizace výkonu a případné paralelizace aplikace (více viz kapitola **Funkcionální návrh a jeho výhody**).

Jádro

Vlastní činností jádra je samotná kompozice výsledného rytmu. Typicky lze průběh algoritmu popsat v několika fázích:

1. Výběr podmnožiny základních rytmů z banky na základě aplikace filtru
2. *Samplování* jednoho nebo více rytmů z podmnožiny
3. Kreativní část algoritmu - typicky se během ní opakují kroky 1 a 2 a získávají další stavení kameny
4. Sestavení sekvence pro danou stopu z kamenů, vybraných v opakovaných krocích 1 a 2

Na základě své specifické vnitřní logiky se každé jádro ve fázi výběru rytmu rozhoduje o parametrech, které by měla mít sekvence, která bude na dané, právě zpracovávané, místo dosazena. Takto specifikované filtrační parametry (korespondující s evaluovanými hodnotami daného rytmu v bance rytmů) jsou potom předány sekvencéru, který jádru předkládá kolekce rytmů (z banky rytmů), které daným filtračním specifikacím vyhovují. Jádro již potom na základě své vnitřní implementace rozhodne, jakým způsobem z nabízených rytmů vybere ten, který v

dané iteraci použije (ať už pseudonáhodně, s přihlédnutím k hodnotě určitého parametru, nebo zcela jiným způsobem).

Jako výsledná délka výstupní sekvence byla zvolena jako nejkratší možná délka, při které může být sekvence složena v podobě písňového formátu (ABAB či ABCB, čehož je využito v implementaci jednotlivých jader - viz kapitola **Navržené varianty jader**). Pokud tedy skládáme sekvenci ze základních kamenů, kde každý trvá čtyři čtvrťové doby (jedna doba celá), je výsledná sekvence dlouhá čtyři takty - a rozhraní sekvencéru tedy od jádra očekává seřazenou množinu o čtyřech prvcích.

Všechny tyto konstanty a vlastnosti systému jsou pak ovládány konfigurací - je tedy možné pouhou změnou konfigurace a výměnou sady základních stavebních kamenů sestavovat například sekvence v tříčtvrťovém taktu o délce osm taktů.

Navržené varianty jader

V rámci této práce je implementována řada jader - kreativních algoritmů, předvádějící možnosti využití navrženého frameworku. V uživatelském testování jsou potom tyto algoritmy jader proti sobě postaveny v rámci tournamentového duelu (kapitola **Testování**).

V rámci této práce byla navržena tato jádra:

1. **simplest** - testovací jádro bez vnitřní logiky; vybere náhodně jednu z filtrovaných (vhodných) sekvencí a tu zřetězí na požadovanou délku
2. **first** - první implementované jádro - skládá sekvence v klasickém písňovém formátu ABAB
3. **heater** - jádro first, rozšířené o přihlédnutí k **heating** koeficientu - u nástrojů s příslušným nastavením je B vzor buď prázdným rytmem, a nebo naopak rytmem, vybraným z jiné podmnožiny filtrovaných stavebních kamenů než vzor A (úpravou jednoho z koeficientů)

4. **missing** - varianta jádra **heater** - místo složitější logiky výběru vzoru B však pouze s určitou pravděpodobností (definovanou konfigurací) nahrazuje vzor B prázdným rytmem
5. **mutating** - implementuje mutaci v nástrojových koeficientech pro výběr vzorů - výsledně tedy skládá formát ABCD, kdy je pro výběr každého ze vzorů upraven každý z definovaných koeficientů nástroje
6. **demutating** - variace **mutating** jádra s opačným znaménkem algoritmu mutace koeficientů jako kontrolní jádro pro test signifikance potvrzení hypotézy vlivu náhody na výslednou estetičnost

Aby bylo možné ve zhodnocení práce naměřená data ještě hlouběji analyzovat a vyřknout komplexnější závěr než pouhé potvrzení či vyvrácení úvodního tvrzení, je třeba navržené algoritmy (více viz kapitola **Jádra - kreativní algoritmy**) roztřídit a ohodnotit ve dvou subjektivně vnímaných navržených dimenzích .

obr. 3 - subjektivní rozložení vlastností výsledných algoritmů na osách

Jak je zřejmé z obrázku 3, navržený způsob ohodnocení algoritmů zkoumá dva arbitrární atributy zkoumaného algoritmu - pravidelnost výsledné formy a vliv náhody na sestavení sekvence, kde pravidelnost formy je přímo úměrná počtu unikátních základních stavebních kamenů, ze kterých je sekvence sestavena (tedy jádro **simplest**, skládající stavební kameny do formy AAAA má pravidelnější formu než jádro **first**, skládající kameny do formy ABAB) a kde je vliv náhody přímo úměrný počtu pseudonáhodných jevů, které chod algoritmu ovlivní (tedy jádro **heater**, kde je pseudonáhodným jevem ovlivněna pouze volba vzorku B ve formě ABAB je méně ovlivněno náhodou než jádro **mutating**, kde je pseudonáhodným jevem ovlivněna volba vzorků B, C a D ve formě ABCD).

Vizualizace

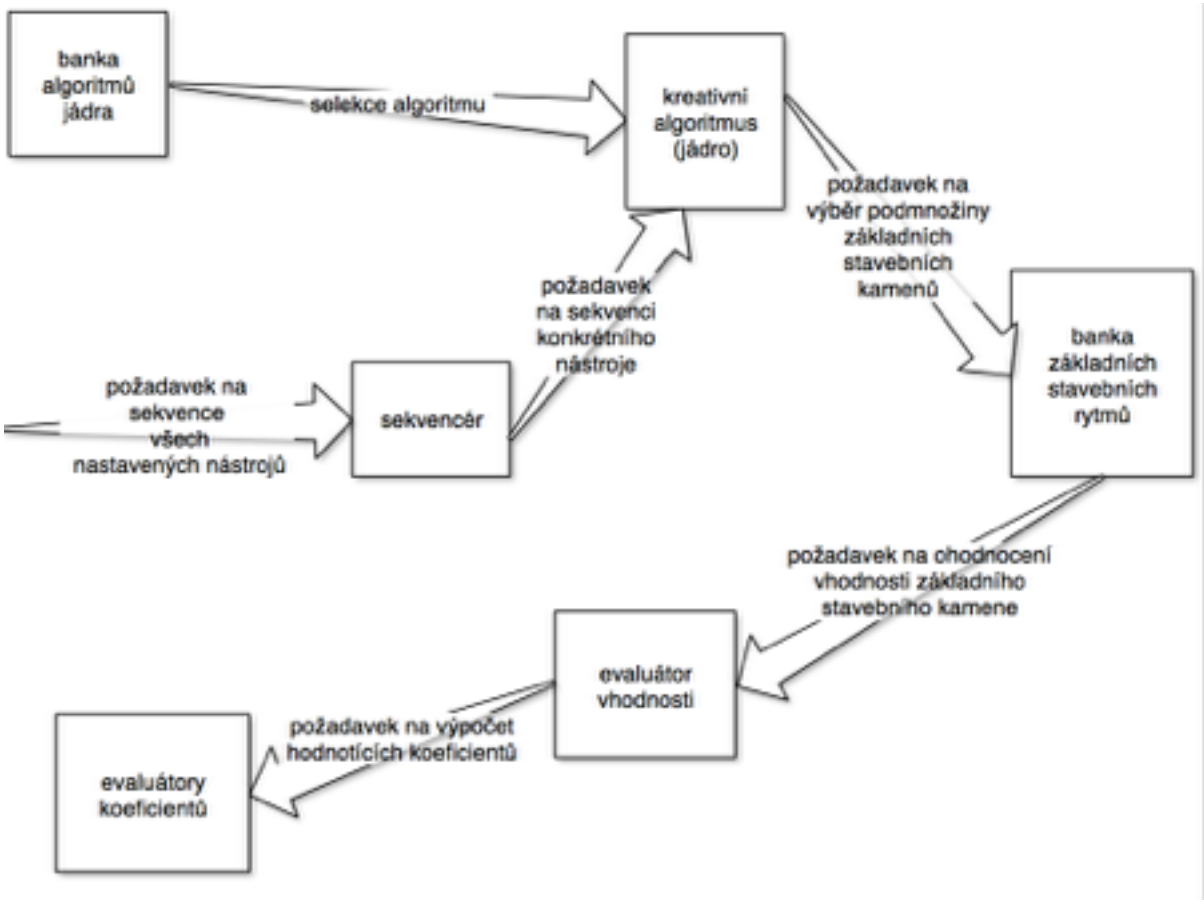
Po kompletním průběhu algoritmu jádra je výsledný rytmus opět serializován a spolu s konfiguračním souborem, definujícím aspekty vizualizace, předán vizualizačnímu modulu. Ten zobrazí výslednou sekvenci namapováním rytmických stop na abstraktní vizualizační nástroje, pro které je v konfiguračním souboru vizualizace specifikována vizuální forma včetně aspektů její reprezentace.

Způsob vizualizace je potom založen na specifikách implementace vizualizačního řešení - pro některé vizualizační platformy bude tedy třeba konvertovat dodaný proud not do vhodnějšího formátu, případně ho rozšířit o informaci, nutnou k vhodné vizualizaci.

Je pak zřejmé, že i navržený způsob vizualizace velmi ovlivní vnímanou míru estetičnosti a případný nevhodný způsob mapování sekvencí na jejich vizuální reprezentace může zcela změnit výsledné uživatelské hodnocení. Vizualizace, implementována v rámci této práce je tedy z velké části založena na subjektivním vnímání autora a je velmi pravděpodobné, že s jinou formou vizualizace by byly výsledky testování a tedy i této práce odlišné.

Architektura a algoritmy

Architekturu projektu je možné chápat z několika úhlů pohledu. Následující přehled je pak sloučením těchto pohledů v podobě reprezentace s důrazem na přirozené konstrukce, definované jazykem a programátorem - s cílem popsat algoritmus z moderního, multiparadigmativního hlediska.



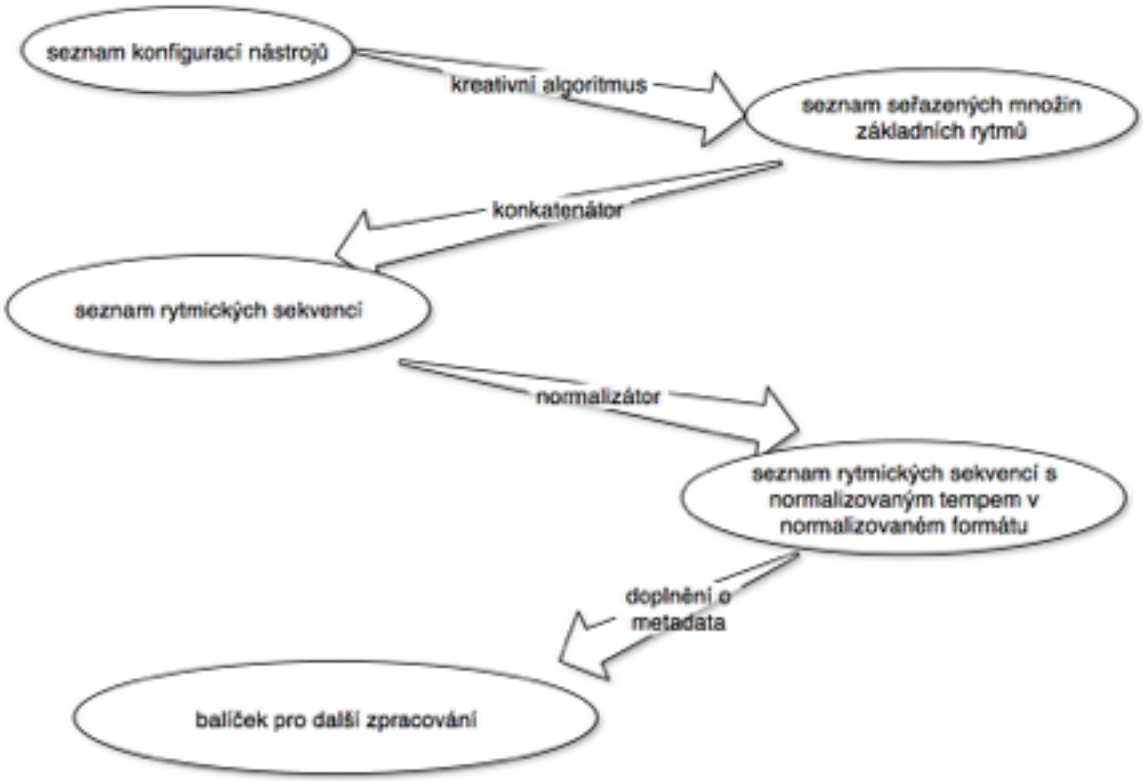
obr. 4 - architektura systému z pohledu vystavených rozhraní

Z obrázku číslo 4 je patrná architektura systému, zjednodušená abstrahováním nebo vynecháním pro architekturu nepodstatných, ale již dříve zmíněných částí řešení - napojení na webový framework, implementace konfigurace specifická pro účely uživatelského testování a podobně. Popis také pro zjednodušení a naplnění formální specifikace vynechává ladící konstrukty (kapitola **Funkcionální návrh a jeho výhody**).

Před samotným během systému je potřeba vybrat z banky algoritmů jádra specifickou instanci, která bude sloužit jako jádro (na základě konfigurace, tournamentového algoritmu či zcela náhodně). Následně pak sekvencér obdrží požadavek na zaslání množiny rytmických sekvencí - reprezentovaný zasláním zprávy, obsahující ukazatel na seznam konfiguračních objektů nástrojů, pro které jsou rytmy vytvářeny (viz kapitola **Návrh řešení a architektury systému,** podkapitola **Konfigurace).**

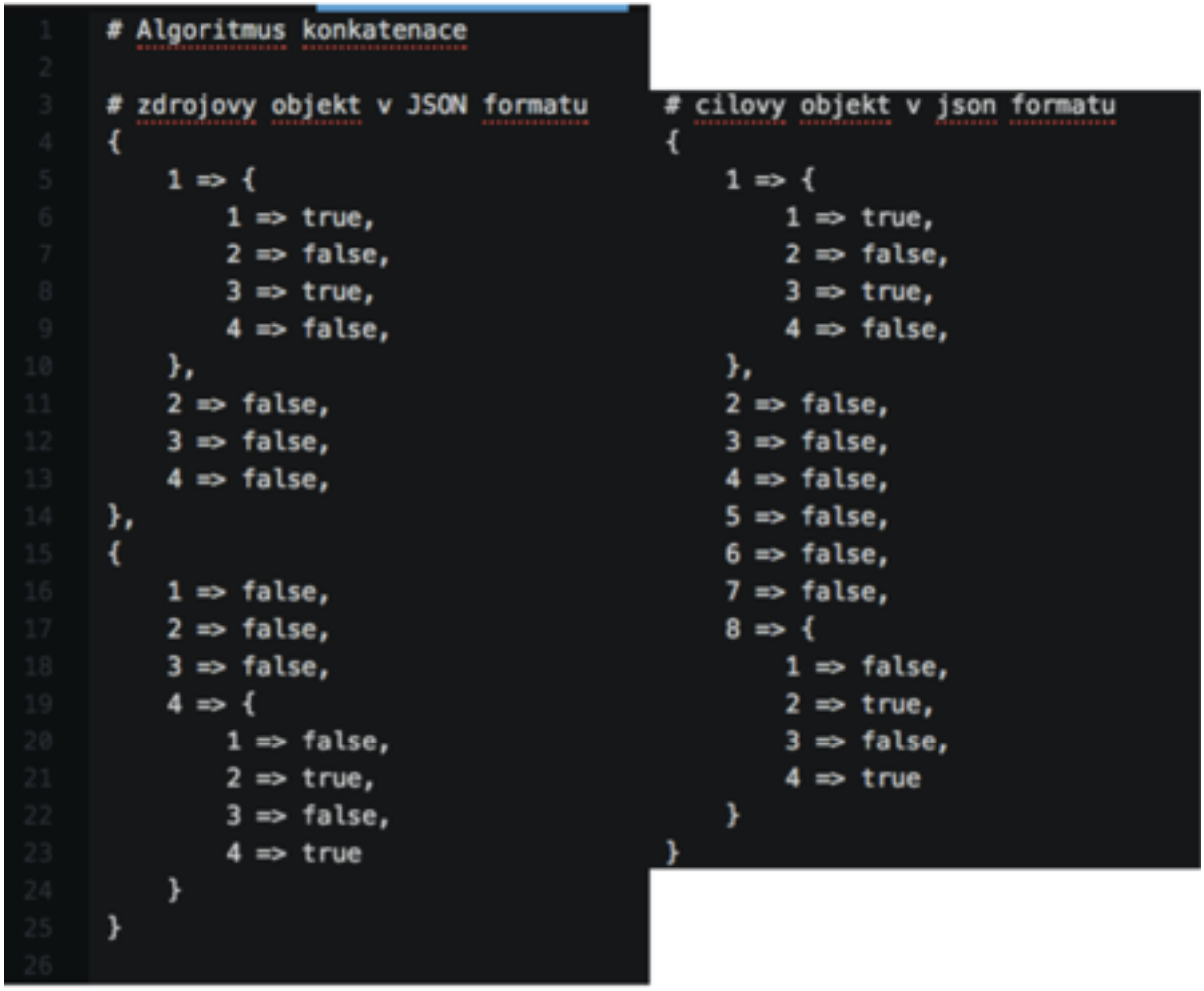
Sekvencér

Sekvencér, implementovaný v podobě funkcionálně navržené pipeline skládá výsledky jednotlivých kreativních algoritmů, které pak postupně skrz řadu filtrů transformuje do podoby, která je vhodná k vizuální a hudební reprezentaci či dalšímu algoritmickému zpracování.



obr. 5 - algoritmus sekvencéru z pohledu datových struktur a jejich transformace

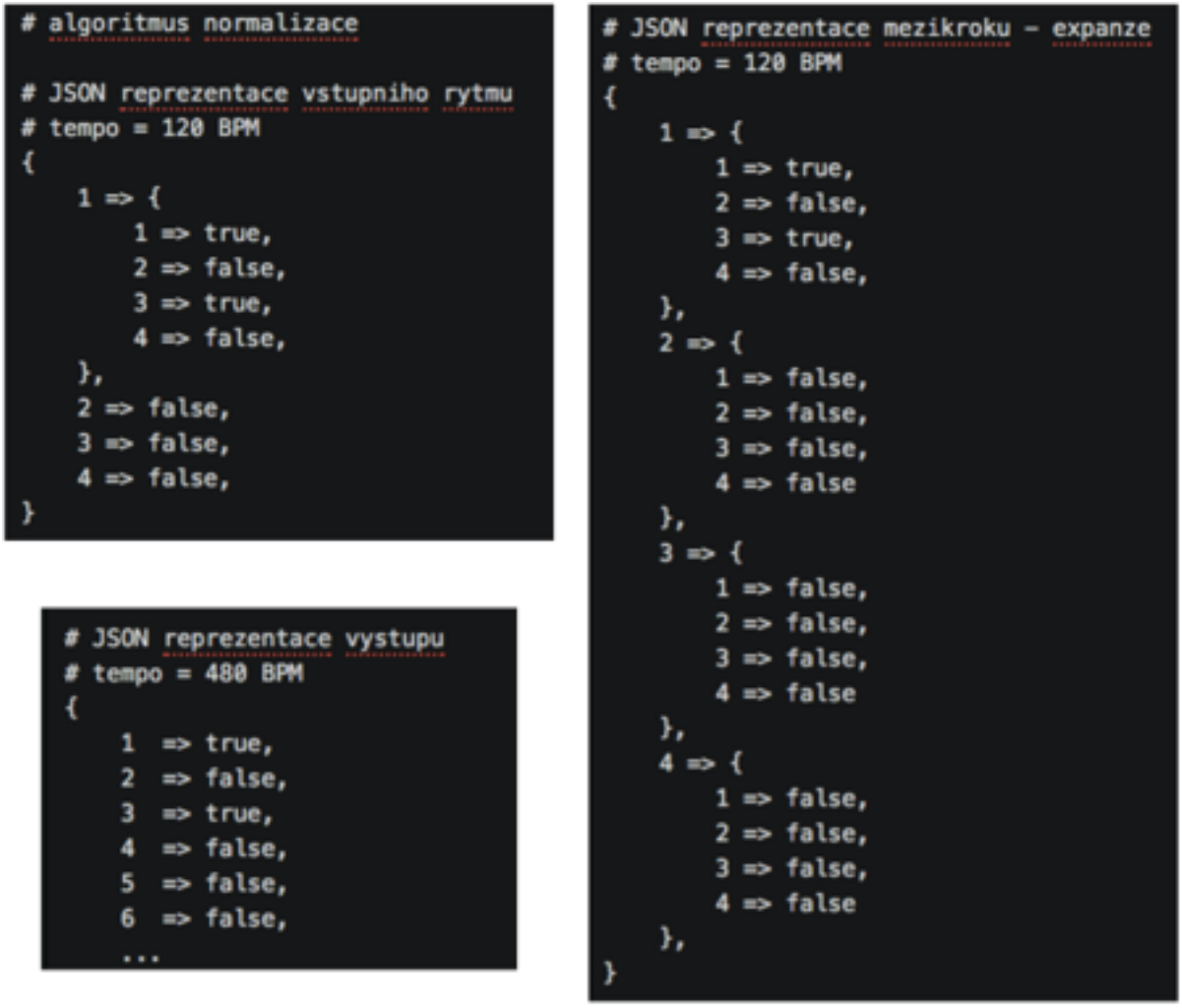
Jak je vidět na obrázku číslo 5, za pomoci seznamu konfigurací nástrojů jako argumentu je volána closure, reprezentující jeden z kreativních algoritmů.
Její hodnotou však je pouze seřazená množina základních rytmů - tu je pak nutné složit do jednoho výsledného objektu, reprezentující konkatenaci rytmu.



obr. 6 - algoritmus konkatenace

Jak je zřejmé z obrázku 6, algoritmus konkatenace silně využívá datové reprezentace rytmu. V případě jiné datové reprezentace by pak složitost a čas implementace narostla - toto je příklad algoritmické mikrooptimalizace, která je v projektu častá - zde je vidět v specializaci datových typů, nad kterými jednotlivé algoritmy pracují zejména pro účely jejich zjednodušení (případ využití principu *preference jednoduchosti nad konvencí*).

Výsledkem algoritmu konkatenace je však formát, který není vhodný k vizualizaci - rytmy jsou stále složené (množina hodnot obsahuje vnořené rytmy). Proto je potřeba nad daty provést operaci normalizace.



obr. 7 - algoritmus normalizace

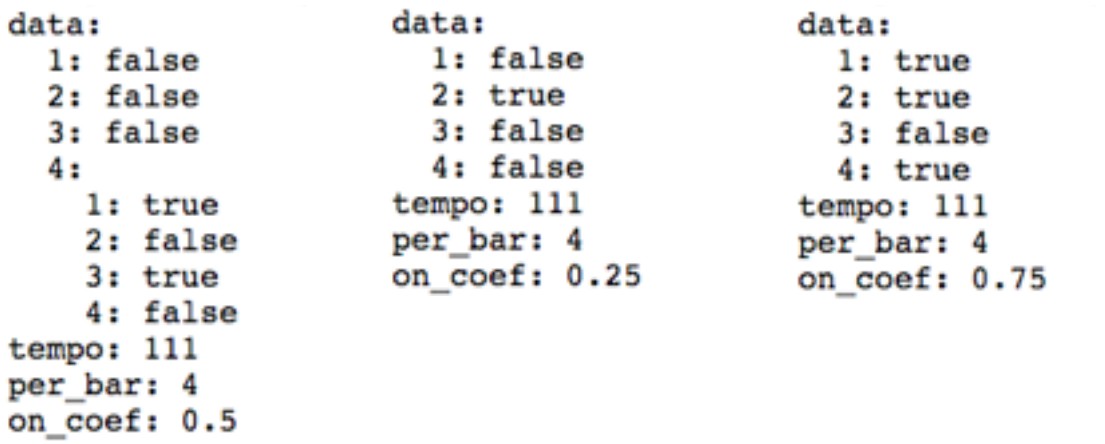
Z obrázku 7 je vidět, že i algoritmus normalizace využívá pro jednoduchost časté transformace datových typů. Rytmus je nejprve rozšířen tak, aby byla jeho struktura uniformní - tedy do takové, kde nejsou delší intervaly než ten nejkratší (a celá reprezentace je tedy složená z not o délce nejkratší noty, která se v původním rytmu objevila) - a následně pouhou rekurzivní iterací transformován do plochého rytmu s násobným tempem, který je pro zobrazování či další práci optimální. Zobrazení usnadní i operace rozšíření rytmu do tempa, která před serializací obdobným způsobem, jako je na obrázku 7 rozšíří všechny rytmické stopy do tempa

nejrychlejšího z rytmů - ve výsledku je pak pro potřeby vizualizace dostačující pouze jedna instance objektu metronomu.

Evaluace rytmů a jejich filtrování

V rámci činnosti kreativního algoritmu (jádra) je klíčovou úvodní fází výběr podmnožiny rytmů, ze kterých potom jádro sestavuje výslednou rytmickou sekvenci. Tato filtrace je interně reprezentována dvěma jevy - ohodnocením základních stavebních kamenů za pomoci výše zmíněné abstrakce kernelové metody - tedy množiny evaluačních funkcí - a následném filtrování této množiny za pomocí aplikace filtračních pravidel, založených na specifikované konfiguraci (připomínající doménově specifický programovací jazyk, ač na této formální specifikaci práce nestaví, viz vysvětlení a zmínka v kapitole **Návrh**).

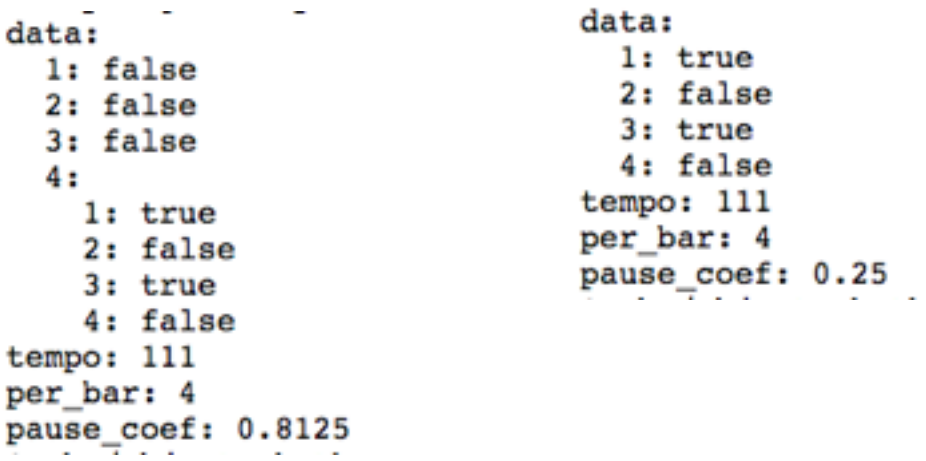
V rámci první fáze jsou nejprve všechny rytmy ohodnoceny podle všech dostupných evaluačních funkcí. Vzhledem k návrhu je možné tuto část předřadit před samotné generování sekvence, a ohodnotit rytmy například už v čase sestavení nebo nasazení do produkčního prostředí. Tato varianta potom zřejmě vede ke zrychlení a optimalizace aplikace, ale zároveň umožňuje i distribuované ohodnocení separovaným subsystémem či příjem evaluací algoritmů jako jeden ze vstupů algoritmu.



obr. 8 - příklad výsledku výpočtu ON koeficientu nad podmnožinou rytmů

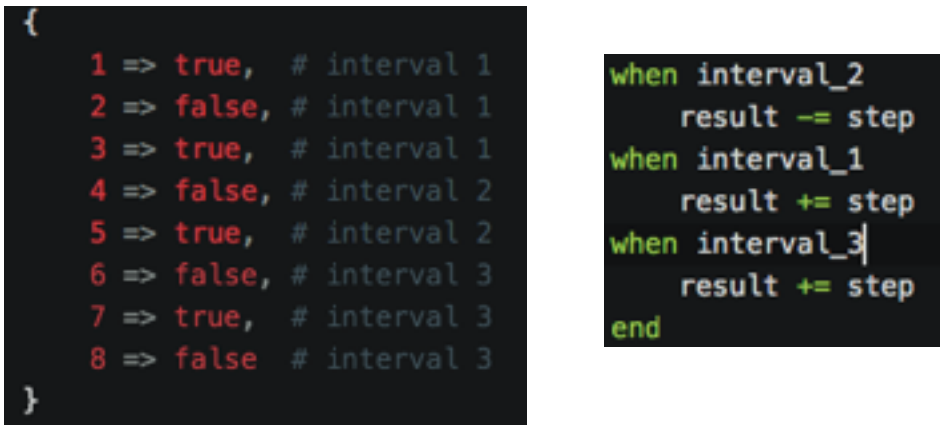
Nejjednodušším z implementovaných hodnotících koeficientů je **ON** koeficient, který popisuje poměr not (v této implementaci boolovské hodnoty *true*) a pomlk (boolovské hodnoty *false*) ploché a sjednocené reprezentace hodnoceného rytmu.

Dalším z implementovaných funkcí evalujících hodnotící koeficient je funkce, sloužící k výpočtu koeficientu **PAUSE**. Tento slouží jako reprezentace nejdelší sekvence pomlk ve výsledné ploché a sjednocené reprezentaci hodnoceného rytmu. Výpočetní funkce bere v potaz fakt, že vizualizace rytmu funguje na bázi zobrazovací smyčky, a tudíž respektuje možnou existenci takové sekvence pomlk, která je vytvořena pouze zobrazením rytmu - tedy takové, kde je první pomlka v sekvenci obsažena v prvním opakování zobrazovaného rytmu a poslední pomlka v sekvenci obsažena v opakování následujícím.



obr. 9 - příklad vypočtených hodnot PAUSE koeficientu

V rámci systému však zvolená implementace koeficientu **PAUSE** přináší hned dvě úskalí. Jednak není průběhem a zvolenou formou průzkumu možných implementací algoritmu jádra zaručeno, že bude každý rytmus zřetězen dalšími instancemi sebe samého (například jádra, skládající písňové formy ABAB - viz kapitola **Navržené varianty jader**), což znamená, že i přes detekovanou řadu pomlk, způsobenou zamýšlenou vizualizační smyčkou nemusí být tato řada pomlk ve výsledné vygenerované sekvenci přítomna. Vzhledem ke zvolené abstrakci však není možné brát na tento fakt zřetel; implementace tedy preferuje přebytečné *false-positive* nálezy jako cenu za minimalizaci *false-negative* nálezů, zejména jako metodu řešící nedostatek v podobě poměrně malého objemu vstupních dat a funkcí, účastnících se ohodnocování rytmů (dimenzí kernelové metody).



obr. 10 - reprezentace chodu algoritmu výpočtu koeficientu FRONT

Posledním navrženým a implementovaným koeficientem je koeficient **FRONT**. Tento vypočítává arbitrární dojmový atribut rytmu - přilehlost not k první době, oproti přilehlosti k prostředku rytmu - druhé a třetí době. Stejně jako **PAUSE** koeficient počítá i tento se zřetězením zobrazní rytmu - a přilehlost ke čtvrté době, která je v zobrazení následována dobou první je tedy zjednodušeně vnímána jako přilehlost k první době. Ač není tato vlastnost univerzální pro celou množinu zobrazitelných rytmů, pro podmnožinu zadaných rytmických sekvencí jde na základě iterativní implementace o dostačující rozdělení. Na rozdíl od **PAUSE** koeficientu zde však není výsledek ovlivněn variací chodu algoritmu jádra - nehledě na to, jestli je za rytmem v zobrazení zřetězen stejný nebo jiný mikrorytmus jde bezpochyby o další čtyřčtvrťový rytmus a jeho jednotlivé doby mají pak stejný sémantický význam.

Technologie

Aspekt volby použité technologie je klíčovou částí implementace, protože značně ovlivňuje empirickou kvalitu výsledného projektu.

Po úvaze a zvážení pro a proti jednotlivých technologií a postupů byla zvolena varianta implementace formou webové aplikace.

Webová aplikace

Webové aplikace jsou v současné době populárním způsobem implementace nápadů. Jejich popularita je založena zejména na jednoduché distribuci - klient takové aplikace je typicky již od výrobce nainstalován ve všech počítačích, tabletech a mobilních telefonech. Vzhledem k rozmachu webových technologií v posledních pár letech má také značná část populace zkušenost s takovou aplikací, zejména díky oblibě jejich typických představitelů, jako jsou sociální sítě, webové vyhledávače či multimediální aplikace **[14]**.

Vyjma snadné distribuce aplikace se popularita této varianty implementace systému projevuje i v dostupných technologických podpůrných systémech. K rychlé tvorbě přenositelné a distribuovatelné aplikace existuje mnoho platforem - například virtualizační platforma Docker **[15]**. I přes úvodní využití jí však nebylo v této konkrétní implementaci využito, hlavně kvůli volbě odlišné proprietární kontejnerové technologie pro nasazení a vydávání aplikace (viz kapitola **Heroku a deployment proces**).

Ruby

Jako implementační jazyk aplikace byl zvolen moderní programovací jazyk Ruby. Ruby je multiparadigmativním (implementuje zejména objektově-orientovaná, funkcionální a imperativní paradigmata) interpretovaným jazykem, který se soustředí zejména na minimalizaci a zpříjemnění programátorovy práce **[16]**. Aspektem, který byl ve fázi výběru technologií klíčovým je potom zejména nízká časová náročnost

případných změn ve fundamentálním aspektu architektury, jež je důsledkem způsobu programování a návrhu, který jazyk Ruby podněcuje **[17]**. Tato vlastnost je vzhledem k explorativnímu způsobu vedení projektu, kdy v úvodu není známo, jaká bude nutná architektura programu, aby bylo možné dosáhnout zamýšlených cílů zcela klíčová - což byl jeden z hlavních důvodů preference jazyku Ruby před konvenčnějšími, rigidnějšími jazyky typu Java či C++.

Funkcionální návrh a jeho výhody

Návazně na současné trendy v optimalizaci algoritmizace je aplikace implementována v duchu funkcionálních paradigmat. Ačkoliv je samotný výsledný kód vytvořen za použití směsi objektově orientovaného a funkcionálního přístupu, jde spíše o pohodlnost při tvorbě či zjednodušení ladění a analýzy aplikace a pokud by byly z kódu sekce, zabývající se laděním vyjmuty, bylo by možné beze změny architektury projektu reimplementovat kód jako čistě funkcionální (například v jazyku Haskell).

```
def evaluate(rhythm)
    rhythm.on_coef = eval_on_coef(rhythm) # kolik % casu je zarizeni zapnuto
    rhythm.front_coef = eval_front_coef(rhythm) # pomer prepnuti v prvni a druhe polovine
    rhythm.pause_coef = eval_pause_coef(rhythm) # jaka je nejdelsi mezera mezi udery?

    return {
        'on' => rhythm.on_coef,
        'front' => rhythm.front_coef,
        'pause' => rhythm.pause_coef
    }
end
```

obr. 11 - příklad metody, implementované za užití směsi funkcionálních a objektově orientovaných paradigmat

Jak je zcela jistě zřejmé, metoda, zobrazena na obrázku 11 není funkcionální - objekt, uložený v proměnné **rhythm** je zcela jistě proměnlivý (*mutable*) a metoda má vedlejší efekty (nejde tedy o čistou matematickou funkci, mapující vstup na výstup).

```
def evaluate(rhythm)
    return {
        'on' => eval_on_coef(rhythm),
        'front' => eval_front_coef(rhythm),
        'pause' => eval_pause_coef(rhythm)
    }
end
```

obr. 12 - výsledek refaktoringu metody z obrázku 11 podle specifikací funkcionálního paradigmatu

Pokud ale z metody odstraníme sekci, zjednodušující ladění programu, dostaneme jako výsledek čistou matematickou funkci - množinu relací vstupů a výstupů, kde je každý jeden člen množiny vstupů mapován na právě jeden člen množiny výstupů.
Taková implementace je potom lépe paralelizovatelná, neboť implicitně je možné každou smyčku, jejíž algoritmus je možné přestrukturovat do podoby ekvivalentního algoritmu popisujícího operaci nad seznamem délky D, který je pak implementován za pomocí funkce vyššího řádu (map, filter a podobné) paralelizovat a to v počtu oddělených vláken či programů N, kde N <= D **[18]**.

Implementace je také lépe testovatelná formou jednotkových (*unit*) testů. Vzhledem k tomu, že je logika tvořená pouze kompozicí bezstavových funkcí se testování obejde bez definic *mock* objektů a jiných způsobů zaobalení testování vedlejších efektů algoritmů.

Framework Ruby on Rails

Jako základ pro implementaci byl použit rámec *Ruby on Rails*. Jde o *open source full stack framework*, určený k implementaci webových aplikací, založený na návrhovém vzoru *Model - View - Controller* a principu *Convention over Configuration* **[19]**. I přes to, že projekt ani zdaleka nevyužívá všechny možnosti frameworku (v projektu je implementován pouze jeden *Controller*, jeden *Model* a jeden *View* - logika je zaobalená do oddělených tříd typu *Service*) se rozhodnutí použít framework ukázalo jako velmi dobré - zejména pro velmi elegantní řešení vstupně-výstupních operací, abstrakci nad vrstvou systému řízení báze dat, podpůrné *scaffoldingové*

skripty či spolupráci se zvoleným produkčním prostředím (více kapitola **Heroku a deployment proces**).

Systém řízení báze dat

V projektu je systém řízení báze dat, implementovaný v podobě *PostgreSQL* serveru pouze podružnou součástí, vzhledem k tomu, že většina konfiguračních dat je uložena v konfiguračních souborech (zejména pro silnější vazbu s kódem a také díky větší pružnosti specifikace konfigurace, které toto řešení přináší) - slouží pouze k uložení informací o výsledku uživatelského testování.

U tohoto případu byl zvolen způsob implementace databází zejména kvůli vhodným nástrojům, které k řešení přináší databázová vrstva *ActiveRecord*, která je typickou součástí aplikace implementované v rámci rámce *Ruby on Rails*. Vyjma dalších modulů, užitých v implementaci (zejména *ORM* mapování) je zde využíván zejména subsystém **databázových migrací** - typicky obsahující seznam záznamů - migrací (každý reprezentovaný samostatným souborem), kde každý popisuje - závisle na implementaci migračního jádra potom buď v jazyce, který slouží k ovládání přímo systému řízení báze dat (např. *MySQL*) či spíše řetězením volání metod nad speciálním **query-builder objektem** (případ *ActiveRecord* implementace v *Ruby on Rails*) - změny v datech či struktuře databáze, které potom příslušné migrační jádro (*engine*) nad nastavenou databází provede.

Toto řešení usnadňuje hned několik problémů - záznam o každém jádře (a jeho skóre v uživatelském testování) je zcela oddělen od kódu a algoritmu jádra - je tedy možné oddělit bez dalších nástrojů akt nasazení nové verze algoritmu od aktu spuštění této verze - zpřístupnění algoritmu publiku. Také dává v kombinaci s verzovacím systémem *git* lepší kontrolu nad daty přidání jednotlivých jader do testu, umožňuje vynulování počítadel skóre v uživatelském testování bez vlivu na kód či přímých, nezaznamenaných zásahů do databáze.

Heroku a deployment proces

Po zevrubné analýze možných řešení byla pro nasazení produkční verze aplikace zvolena platforma *Heroku*, cloudové řešení, založené na soustavě spravovaných kontejnerů (ne nepodobných například zmiňovaným *Docker* kontejnerům - viz podkapitola **Webová aplikace**). Vyjma placených programů umožňuje platforma i hostování neomezeného počtu zkušebních programů zdarma. Vzhledem ke struktuře platformy je pak rozdíl mezi verzí placenou a verzí zdarma zejména ve výkonu, který je jednotlivým programům přiřazen k dispozici. I se zkušební verzí je tedy v současné době možné využívat nástrojů *Heroku toolbelt*, které mimo jiné umožňují například pohodlné nasazení formou události, vyvolané při operaci *push* do větve *master* ve verzovacím systému *git* či přístup k záznamovým souborům či omezené podmnožině utilit virtualizovaného kontejneru **[20]**.

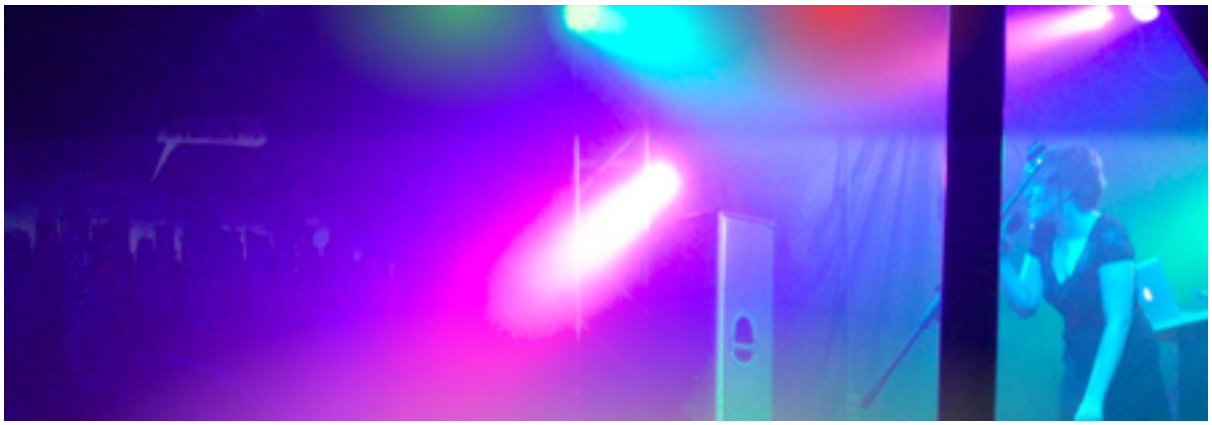
Zejména díky kombinaci nasazení z verzovacího systému *git* a bezúdržbové operativě zvolené platformy s vhodně zvoleným způsobem správy nasazování změn v produktu (viz podkapitola **Systém řízení báze dat**) bylo dosaženo robustního, škálovatelného a v rámci mezí profesionálního řešení s nulovou finanční a velmi nízkou časovou investicí.

Zobrazovací modul

Pro vizualizaci výsledků průběhu algoritmu, jejich subjektivnímu ohodnocení a následnému uživatelskému testování slouží zobrazovací modul. V rámci této práce jde o případ typické implementace rozhraní webové aplikace v podobě kombinace *HTML5* notace pro reprezentaci prvků, *CSS3* stylů a programovacího jazyka *JavaScript* v kombinaci s knihovnou *jQuery* pro *binding* rozhraní s webovou službou.

V rámci implementace však bylo nutné vyřešit jedno specifikum aplikace - a to světelnou animaci, užívanou v rámci vizualizace. Ta je potom ve finále kompozicí několika technik - světla jsou definována v podobě *radiálních gradientů*, světelné přechody pak jako oddělené třídy, které definují délku animace v dané fázi světla

(rozsvěcení a zhasínání). Vizualizační JavaScriptový modul, synchronizovaný za pomoci open-source nástavby *WAAClock* nad standardem *Web Audio API* **[21]** potom jen v závislosti na notách vytvořeného rytmu spouští zobrazovací *callback* funkce daného nástroje, která pak vhodným způsobem přepíná třídy nad daným *DOM objektem*, výsledkem čehož je zobrazení animace rytmu.



obr. 13 - snímek zobrazované animace

Implementace

Možných přístupů k implementaci aplikace je hned několik. Aby bylo možné informovaně zvolit způsob a dráhu implementace, bylo nejprve nutné definovat očekávané vlastnosti a metriky, které nejlépe reflektují zamýšlené použití aplikace.

Výsledná aplikace by měla být zejména **pružná** a implementována za pomocí **agilních metodik [12]** - případná změna ve specifikaci (což je u projektů, kde je cíle dosahováno zejména *breadth-first* průzkumem možného pole řešení častý jev) by tedy neměla způsobit značnou změnu v kódové bázi projektu.

Dalším aspektem aplikace, který napomáhá pružné reakci na případné změny ve specifikaci či záměru nebo užití aplikace je **modularita.** Cílem je stav kódové báze, kdy i značná změna případů užití (například rozhodnutí, že vizualizace aplikace bude implementována v *embedded* zařízení za účelem aplikace výsledků

kreativního algoritmu na reálný kus osvětlení či elektronického hudebního nástroje) znamená pouze minimální nebo zanedbatelný zásah do výsledné architektury. Tohoto je dosaženo důsledným rozdělením funkcionalit do samostatných modulů s přesně definovaným polem působnosti a rigidními vzájemnými rozhraními. Výsledná aplikace potom není monolitický a obtížně analyzovatelný celek, ale spíše abstraktní *pipeline*, složená z jednotlivých dílčích modulů.

Důležitým atributem aplikace je také její **konfigurovatelnost** - vzhledem k případným variantám aplikace je klíčové rozdělení aplikace do třech základních struktur - **framework, business logiku** a **konfiguraci.**

Framework

Největší částí implementace je vytvořený **framework**. Z abstraktního pohledu jde o skupinu knihoven, spojených metodikou *loose coupling, high cohesion* **[13]**. Tato množina knihoven slouží k abstrahování příslušných typických operací - vstup a výstup, zaobalení a abstrakce práce se systémem řízení báze dat - ale i operací, specifických pro toto zadání - zejména operace nad objekty rytmů, jejich slučování a morfózu do jiných délek taktu. Cílem bylo vytvořit takový framework, kterého mohou jednotlivé navržené kreativní algoritmy využívat k dosažení zamýšleného cíle se záměrem neporušit abstrakci.

Business logika - kreativní algoritmy

Klíčovou součástí většiny aplikací je *business logika* - v tomto případě reprezentována několika variantami kreativních algoritmů, skládajících výslednou rytmickou sekvenci - jader. Vzhledem k záměru testovat několik implementací jader a porovnávat jejich vnímané vzory a specifika bylo cílem vytvořit takovou reprezentaci jádra, která bude vhodně zaobalená a abstrahovatelná, a bude mít jasně definované rigidní rozhraní. Důsledkem tohoto rozhodnutí je fakt, že je v konkrétní implementaci možno bez změny kódu frameworku zapojit automaticky libovolné z implementovaných jader.

Jako zvolený způsob reprezentace byla vybrána metoda reprezentací přes closure - tedy (zpravidla) matematických funkcí sloužící jako oddělený jmený prostor. Taková reprezentace slouží nejenom k plnému uspokojení požadavků na architekturu systému, specifikovaných v předcházejících kapitolách, ale také umožňuje další rozšíření aplikace za užití pokročilých paradigmatů a technik funkcionálního programování, zjednodušuje implicitní paralelizaci algoritmu a usnadňuje testování základních celků.

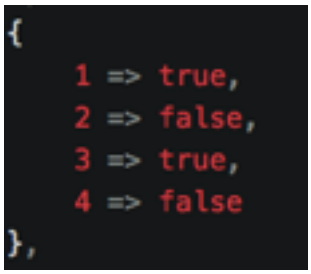
Konfigurace

Záměrem zvoleného způsobu implementace práce je co největší možná konfigurovatelnost výsledného produktu - tedy možnosti změnit funkcionalitu nebo její specifika úpravou konfiguračního souboru (ten tedy může být například přijímán jako jeden z uživatelských vstupů pro personalizaci aplikace - přizpůsobení jejího chodu či vzhledu na základě uživatelovy osobní konfigurace) bez nutnosti účasti programátora. Toto výrazně zvyšuje variabilitu výsledku; a přímo vybízí k dílům navazujícím na tuto práci, zkoumající vliv odlišné konfigurace na zjištěný výsledek.

V této konkrétní implementaci jde o dvě oddělené konfigurace - konfiguraci základních rytmů, ze kterých algoritmus komponuje sekvence, a konfiguraci jednotlivých nástrojů - parametrů, které ovlivňují činnost jednotlivých výpočetních jader.

Základní stavební kameny

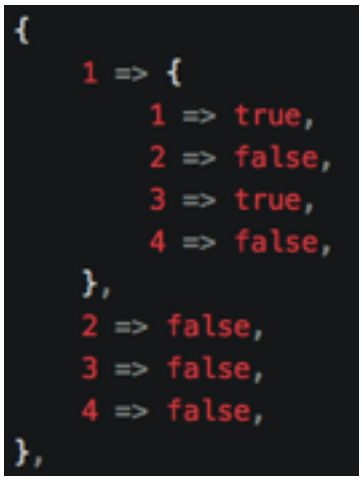
Na základě úvahy (viz kapitola **Návrh řešení a architektury systému**, podkapitola **Banka rytmů**) slouží jako množina základních stavebních kamenů pro algoritmy jader banka základních stavebních kamenů. Jde o seznam (seřazenou množinu) jednotlivých rytmů - kde každý rytmus je definován jako uspořádaná množina dvojic či relací klíč - hodnota (v některých terminologiích a programovacích jazycích “asociativní pole”), ve které klíče nabývají hodnot celých čísel 1 až 4, a hodnoty jsou proměnnou boolovského typu (pravda či nepravda).



obr. 14 - příklad reprezentace jednoduchého základního stavebního kamene

V navrženém datovém modelu pak klíče odpovídají dobám, případně při dalším dělení zlomkům dob - a případná pravdivá boolovská hodnota odpovídá úderu (notě) v tomto čase. Každá doba v neděleném rytmu pak odpovídá v reprezentaci jedné čtvrtinové notě či pomlce a celý základní stavební kámen pak reprezentuje jeden takt.

Vzhledem k tomu, že takto zjednodušená reprezentace rytmu nestačí k uspokojivému generování odlišných estetických sekvencí byla v průběhu implementace definice rozšířena o tzv. “složené rytmy” - kde hodnotou v páru relace může krom proměnné boolovského typu i další rytmus, tedy uspořádaná množina dvojic.



obr. 15 - příklad reprezentace složeného základního rytmu

Tato reprezentace byla zvolena zejména kvůli optimalizaci hodnotících algoritmů - pro část z nich znamená zjednodušení jejich fungování a umožňuje jejich rekurzivní implementaci - ale pro jiné moduly (zejména vizualizační) musí být převedena do jednodušší formy (viz kapitola **Sekvencér**). Pro zanořené hodnoty rytmů (na

obrázku 3 tedy hodnota první doby základního rytmu) pak platí pravidlo, že zanořený rytmus má jako celek délku takovou, jakou by měla nota či pomlka, na jejímž místě se nachází. Vnořené doby pak v tomto případě reprezentují noty či pomlky o délce jedné šestnáctiny taktu.

Uživatelské testování

Vzhledem k zadání a směru práce, jakož i úvodní definici pojmu *estetika* a jeho vnímaného významu je důležitou součástí potvrzení zkoumaných hypotéz za pomocí navrženého a implementovaného systému. V rámci této implementace je celé testování obsažené v rozšířené implementaci zobrazovacího modulu a několika konfiguracích, specifikujících chování systému.

Navržená metodika

V rámci potvrzení nebo vyvrácení počátečního tvrzení **Je možné generovat unikátní rytmické sekvence, které člověk vnímá jako estetické** je nutné zahrnout v projektu i fázi testování. To samotné bude probíhat ve dvou fázích - první, zkušební, kde testovacím subjektem bude pouze autor aplikace, a druhé, kde subjektem budou náhodní a ochotní uživatelé a návštěvníci výsledné webové aplikace, vyhledatelné pod doménovým jménem **bakalarka.herokuapp.com.**

V kontextu zvolených technologií, cílové aplikace i velikosti množiny dobrovolníků, která je v rámci testování k dispozici je v implementaci testovacího subsystému kladen důraz zejména na simplifikaci a minimalizaci nutného vstupu uživatele, ať už datového, myšlenkového či časového.

Na základě tohoto rozhodnutí byla pro provedení testování navržena varianta tournamentového systému v podobě duelu. V ní jsou každému návštěvníkovi webové aplikace předloženy dvě instance vizualizací dvou rytmů s jednoduchou otázkou - “Který rytmus je hezčí?”.

Toto řešení minimalizuje nutný vstup od uživatele, standardizuje ho do nejjednodušší možné reprezentace a řeší specifika systému, například vyhnutí se vágním vstupům typu hodnocení na stupnici od 1 do 7 (v rámci testu webovou aplikací není bez dalšího systému nebo úpravy testování zřejmé, kolik hodnocení uživatel provedl - není tedy jasné, jestli jeho hodnocení rytmu číslovkou 5 značí, že se mu rytmus poměrně líbí či že je rytmus nadprůměrně kvalitní oproti dalším, které viděl) či obtížně analyzovatelným textovým vstupům.

Zamýšleným výsledkem testování je tedy jedna procentuelní hodnota pro každé zkoumané jádro - a to poměr výher v celkovém počtu duelů, kterých se jádro účastnilo.

Pro co největší simplifikaci výběru jader pro účast v duelu je dvojice jader vybrána vždy zcela náhodně, což zřejmě při dostatečném počtu duelů vyústí v rozložení, konvergující k uniformnímu rozložení.

Pro zjednodušení testování v první (zkušební) fázi je připravena také heuristika, napomáhající výběru nově přidaného jádra - v ní má prioritu účasti v duelu to jádro, které se účastnilo nejnižšího počtu klání (jeho soupeř je vybrán náhodně).

Naměřená data a jejich význam

V rámci testování, probíhajícím od 9. dubna 2016 do 7. května 2016 (28 dnů) se podařilo nasbírat celkem 614 ohodnocení.

jádro	výher	proher	duelů celkem	procento výher
heater	60	45	105	57,1428571428571
mutating	36	65	101	35,6435643564356
demutating	44	57	101	43,5643564356436
first	57	44	101	56,4356435643564
missing	52	49	101	51,4851485148515
simplest	58	47	105	55,2380952380952
CELKEM	307	307	614	

obr. 16 - výsledky uživatelského testování

Podle očekávání se až na výjimku (jádro **mutating**) pohybují procentuální poměry výher všech jader pod 10% odchylky od uniformního rozložení výher. I bez statistických operací je zřejmé, že výsledná signifikance získaných dat je velmi nízká a není tedy možno přesvědčivě tvrdit, zda některé z jader generuje obecně estetičtější sekvence.

Lze také tvrdit, že je pro snížení chyb nutné provést výrazně větší počet měření, protože stávající výsledek může být ovlivněn nedokonalostí pseudonáhodného výběru, který je v procesu přítomen hned na několika místech (tvorba sekvence, výběr jádra..).

Přesnost měření lze také zlepšit hlubším záznamem každého hodnocení - například se zohledněním přiřazené *session* hodnoty či řetězce *user agent* - a následné analýze zdrojů získaných dat. Implementovaná simplistická metoda testování však se sběrem takových dat nepočítá, a je tedy možné, že je měření zkresleno velkým počtem hodnot, vytvořených jednou osobou - a výsledná data potom spíše reflektují jeho či její subjektivní pohled.

Je také pravděpodobné, že výsledek reflektuje neideální přístup k návrhu variant algoritmů jader - zjistili jsme jen, že jejich výsledky jsou srovnatelné, ale nevíme, zda jsou dobré nebo špatné.

Výsledek také může být ovlivněn formou vizualizace, ať už grafickým stylem či celkovým provedením - a že s jinou formou, například kombinující audio a vizuální prvky je možné dojít k jednoznačnějším závěrům.

jádro	vliv náhody	pravidelnost formy	procento výher
heater	2	3	57,14285714
mutating	4	2	35,64356436
demutating	4	2	43,56435644
first	1	4	56,43564356
missing	3	4	51,48514851
simplest	1	5	55,23809524

pearsonův korelační koeficient

náhoda - výhry	pravidelnost - výhry
-0,8696249789	0,7559198317

obr. 17 - Pearsonův korelační koeficient

Pokud ovšem vezmeme v potaz arbitrární ohodnocení jednotlivých implementovaných jader (viz kapitola **Navržené varianty jader**) a vypočteme Pearsonův korelační koeficient mezi tímto arbitrárním ohodnocením, vidíme, že vliv náhody zřejmě negativně koreluje s úspěšností jádra (tedy **čím menší vliv pseudonáhody na tvorbu sekvence, tím je algoritmus úspěšnější**) a že vliv pravidelnosti formy zřetelně pozitivně koreluje s úspěšností jádra (tedy **čím pravidelnější je forma, tím je algoritmus úspěšnější**).

Jak je vidět na obrázku 7, vypočtená hodnota korelačního koeficientu má nižší váhu, než se může na první pohled zdát - lze jen obtížně říct, že má jádro **missing** třikrát větší vliv náhody než jádro **simplest** - vliv náhody je zřejmě výrazně vyšší, ale ohodnocení číslicí 3 je arbitrární.

Ač je možné pochybovat o míře korelace, je zřejmé, že se v naměřených datech projevuje a pomáhá nám lépe poznat a analyzovat kreativní procesy a jejich výsledky.

Závěr

V rámci práce byl na základě načerpaných vědomostí z literatury a za použití moderních softwarových metodik navržen a implementován robustní modulární framework, určený pro implementaci algoritmů - jader, generujících rytmické sekvence za pomocí techniky kompozice.

Nad vytvořeným frameworkem byla pak vystavěna webová aplikace, určená k ověření vyřčeného tvrzení **Je možné generovat unikátní rytmické sekvence, které člověk vnímá jako estetické** za pomoci sady implementovaných jader. Aplikace získávala formou uživatelského testu, kdy se turnajovým způsobem utkávaly dvojice rytmických sekvencí, vygenerovaných různými jádry s jednoduchou otázkou, očekávající uživatelův vstup - **Který rytmus je hezčí?**

Na základě restrikce zadání byly porovnávané algoritmy implementovány bez užití *knowledge based* algoritmů, jako jsou umělé neuronové sítě či jiné implementace strojového učení. Toto umožnilo analýzu a klasifikaci navržených prototypů algoritmů jader, na základě kterého byla vytvořena testovací sada, zkoumající vliv náhody a formy na výslednou estetickou hodnotu.

I přes to, že na základě získaných dat nebylo možné potvrdit nebo vyvrátit vyřčené úvodní tvrzení byla odhalena zjevná korelace formy a míry náhody s vnímanou estetičností generovaných estetických sekvencí. K dalšímu zkoumání problematiky také vybízí robustní konfigurace, ovlivňující chod implementovaného systému, a je tedy možné, že konečný závěr nad pravdivostí úvodního tvrzení bude vyřčen až v rámci případné práce navazující.

Zdroje

[1] Smith, Reid. "Knowledge-Based Systems Concepts, Techniques, Examples". http://www.reidgsmith.com. Schlumberger-Doll Research

[2] Cestrová, Monika. “Jan Mukařovský a jeho pojetí estetična”, Ústav estetiky Filozofické fakulty Jihočeské univerzity v Českých Budějovicích, 2009

[3] Mukařovský, Jan: Estetická funkce, norma a hodnota jako sociální fakty, in: Studie I., Brno: Host, 2007

[4] Image aesthetic evaluation using paralleled deep convolution neural network, Lihua G., Li F., School of Electronic and Information Engineering, South China University of Technology, 2015

[5] RAPID: Rating Pictorial Aesthetics using Deep Learning, Lu X., Hailin Jin Z., Yang J., Wang J. Z., Pennsylvania State University, 2014

[6] THE ASSOCIATIVE BASIS OF THE CREATIVE PROCESS, Mednick A.S, University of Michigan, 1962

[7] GENERATING SEQUENCES WITH RECURRENT NEURAL NETWORKS, Graves A., University of Toronto, 2013

[8] THE EUCLIDEAN ALGORITHM GENERATES TRADITIONAL MUSICAL RHYTHMS, Toussaint G., McGill University Montreál Québec, 2005

[11] An introduction to Support Vector Machines and other kernel-based learning methods, Cristianini N., Shawe-Taylor F., Cambridge University Press New York 1999

[12] The agile manifesto, Fowler M., Highsmith J., Software Development 2001 http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf

[13] Modularity, flexibility and knowledge management in product and organization design, Sanchez R., Mahoney J.T., Strategic management journal, 1996

[14] Mislove, Alan, et al. "Measurement and analysis of online social networks." Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. ACM, 2007.
[18] Lämmel, Ralf. "Google's MapReduce programming model—Revisited." Science of computer programming 70.1 (2008)

Dodatečné zdroje

[9] Slicex plugin pro FL studio, Image-Line Software https://www.image-line.com/support/FLHelp/html/plugins/Slicex.htm

[10] Arpeggiator plugin pro FL studio, Image-Line Software https://www.image-line.com/support/FLHelp/html/pianoroll_arpeggiate.htm

[15] What is Docker, https://www.docker.com/what-docker

[16] The Ruby Community: About Ruby, https://www.ruby-lang.org/en/about/

[17] LEGOs, Play-Doh and Programming, Jamis Buck, 2008 http://weblog.jamisbuck.org/2008/11/9/legos-play-doh-and-programming

[19] The Rails doctrine. http://rubyonrails.org/doctrine/

[20] The Heroku platform. https://www.heroku.com/platform

[21] Web Audio API W3C Working Draft. http://www.w3.org/TR/webaudio/