Akademie múzických umění v Praze

Hudební a taneční fakulta

Hudební umění

Skladba

DIPLOMOVÁ PRÁCE

Virtuální prostor jako interaktivní hudební kompozice: generování hudby ve videohrách na základě analýzy obrazu

Jan Rösner

Vedoucí práce: doc. MgA. Jan Trojan, Ph.D.

Přidělovaný akademický titul: MgA.

Praha, červen 2023

The Academy of Performing Arts in Prague
Music and Dance Faculty

Art of Music
Composition

MASTER'S THESIS

Virtual space as an interactive music composition: generating music in video games based on image analysis

Jan Rösner

Thesis supervisor: doc. MgA. Jan Trojan, Ph.D.
Awarded academic title: MgA.

Prague, June 2023

P r o h l á š e n í

Prohlašuji, že jsem magisterskou práci s názvem

Virtuální prostor jako interaktivní hudební kompozice: generování hudby ve videohrách na základě analýzy obrazu

vypracoval samostatně pod odborným vedením vedoucího práce a s použitím pouze uvedené literatury a pramenů a že práce nebyla využita v rámci jiného vysokoškolského studia či k získání jiného nebo stejného titulu. Souhlasím s tím, aby práce byla zveřejněna v souladu se zákonem a vnitřními předpisy AMU.

Praha, dne ……. ……..

Poděkování

Rád bych poděkoval Janu Trojanovi za dlouhodobou konzultaci celého projektu, kterému se tato práce věnuje. Také děkuji Jakubovi Krejčímu, který pomáhal vytvářet první verzi projektu, a navedl mě na řešení analýzy obrazu. Na závěr chtěl poděkoval Kryštofu Ježkovi a Vendule Burešové, se kterými jsem spolupracoval v další verzi projektu.

Abstrakt

Diplomová práce se zaměřuje na experimentální využití generativní hudby ve videohrách. Popisuje koncepty pro generování hudby v reálném čase na základě analýzy obrazu videohry. Příprava na práci zahrnovala dvouletý projekt, který měl za cíl vytvořit experimentální počítačovou hru s generativní hudbou založenou na analýze obrazu. I když se nepodařilo dosáhnout plně funkčního stavu projektu, experiment přinesl částečné poznatky, které jsou diskutovány a reflektovány v práci. Získané poznatky z experimentu slouží jako inspirace pro další vývoj a implementaci tohoto konceptu. Práce se soustředí také na specifika videoherní hudby a provádí srovnání s hudbou filmovou. Podrobněji popisuje technologické aspekty videoherní hudby relevantní pro projekt. Dále se zaměřuje na obecnou problematiku generativní hudby ve videohrách.

Klíčová slova: generativní hudba, videohry, hudba pro videohry, adaptivní hudba, interaktivní hudba

Abstract

This thesis focuses on the experimental use of generative music in video games. It describes concepts for generating music based on real-time analysis of the game's visuals. The preparation for this work involved a two-year project aimed at creating an experimental computer game with generative music based on image analysis. Although the project did not achieve full functionality, the experiment yielded partial insights that are discussed and reflected upon in the thesis. The findings serve as inspiration for further development and implementation of this concept. The thesis also delves into the specifics of video game music, comparing it to film music, and provides a detailed exploration of the relevant technological aspects of video game music for the project. Additionally, it addresses the general issues surrounding generative music in video games.

Keywords: generative music, video games, video games music, adaptive music, interactive music

Obsah

Úvod

Jako příprava na tuto práci vznikal po dobu dvou let projekt, jehož cílem bylo zrealizovat experimentální počítačovou hru, ve které je hudba generovaná v reálném čase na základě analýzy obrazu hry. Ačkoliv se tento projekt z technických důvodů nepodařilo uvést do funkčního stavu, a tak nebylo možné prakticky vyzkoušet nápady a umělecké postupy zde popsané, některé dílčí poznatky tento experiment přinesl. Navržená řešení určitých problémů, které generování hudby k videohře na základě analýzy obrazu přináší, jsou dle autorova názoru stále platná. Také určité části designu navrženého řešení mohou posloužit jako inspirace pro další pokus o realizaci tohoto experimentu. Přínosné jsou rovněž poznatky o tom, jaké technické řešení se zdá pro tento experiment nevhodné – jinými slovy to, proč se projekt nepodařilo uvést do funkčního stavu.

Nápad vytvořit experimentální videohru, která funguje jako interaktivní hudební kompozice, vznikl při otázce, jakým způsobem hudebně popsat prázdnost (nevybavenost) obytného prostoru. Tato otázka se zpočátku vztahovala ke konkrétní místnosti, která samozřejmě nebyla zcela prázdná – nacházel se v ní stůl, skříň, a další objekty, které by člověk v takovém pokoji očekával; jen těch objektů bylo o něco méně, než je obvyklé. Při delší úvaze nad touto otázkou se ukázala jako zásadní myšlenka, že není podstatná faktická přítomnost, či v tomto případě spíše absence objektů v prostoru, ale samotný vizuální vjem. To přirozeně vedlo k rozhodnutí vytvořit videohru ve 3D.

Tvorba hudby pro videohry je v celé řadě ohledů velmi specifická. Práce tedy začíná srovnáním hudby pro videohry a dalším typem užité hudby – hudbou filmovou. Poté jsou v práci představeny a vysvětleny často užité pojmy a techniky, se kterými se lze v tomto médiu na poli hudby setkat. Vůči nim se také vymezuje generativní přístup k hudbě, který přináší nová řešení, ale i problémy. Ty jsou dále rozebrány v samostatné kapitole věnované generativní hudbě ve videohrách obecně. V závěrečné kapitole je rozebrán výše zmíněný experiment.

Vzhledem k tomu, že práce rozsáhle pojednává o videohrách, je pojem videoher, který jinak zahrnuje počítačové, konzolové, ale i historické „arkádové“ hry, zkrácen pouze na „hry“.

1. Specifika herní hudba

V této kapitole jsou popsána specifika herní hudby. Koncepty a již existující technická řešení jsou důležitá pro porozumění hlavnímu tématu práce: využití dat z analýzy obrazu hry ke generování hudby v reálném čase.

1.1 Srovnání filmové a herní hudby

Ačkoliv jde v obou případech o typ užité hudby, v mnoha rysech se liší. Cílem této kapitoly není najít vyčerpávající seznam rozdílů mezi filmovou a herní hudbou, ale poukázat na podstatné odlišnosti, které jsou dále významné pro herní hudbu.

1.1.1 (Ne)linearita

Asi tím nejvýraznějším rozdílem mezi filmovou a herní hudbou je linearita. Zatímco filmová hudba je vždy ve výsledku napsána pro fixní médium, a je tedy lineární, hra je médium dynamické a nelineární. Rozhodnutí hráče mají vliv na tempo průběhu samotné hry, a tedy i toho, jaká hudba má zrovna znít. Například, rozhodne-li se hráč jít do jiné lokace ve hře, musí na to reagovat i hudba. Další běžnou situací ve hrách bývá nepředvídatelná proměna stavu hry, např. nenadálé započetí souboje. Pro tyto účely vznikla řada různých technických řešení (viz 1.3 Technický přístup k herní hudbě), které lze shrnout pod pojem adaptivní hudba. S touto proměnlivostí se rovněž může vypořádat i generativní přístup k hudbě.[1] [2]

V některých případech se lze setkat také se situacemi, ve kterých se herní hudba nachází v identické situaci k hudbě filmové. To bývají tzv. cut-scény, což jsou neinteraktivní a neměnné scény v počítačové grafice, které zpravidla slouží k posunutí děje. Tyto scény jsou ze své podstaty lineární, a tak umožňují kompozici soundtracku přímo na míru konkrétní scéně.

1.1.2 Role herní hudby

Herní hudba má v řadě případů podobnou úlohu jako hudba filmová. Tyto role je možné shrnout v několika bodech, které se ale do určité míry překrývají:[3]

[1] Více o nelinearitě ve videohrách viz:
COLLINS, Karen. Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design. Cambridge, Mass: MIT Press, 2008. ISBN 978-0-262-03378-7. s. 142–147.

[2] BERNDT, Axel. Musical Nonlinearity in Interactive Narrative Environments. In: *Proceedings of the International Computer Music Conference* [online]. August 16–29, 2009, Montreal, Kanada. Dostupné z: https://www.researchgate.net/publication/261803850_Musical_Nonlinearity_in_Interactive_Narrative_Environments/references
[cit. 20-6-2023]

[3] SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7.
V této knize jsou uvedeny stejné body. Zde uvedená formulace se v některých místech liší, a především uvádí příklady, které zná autor z vlastní zkušenosti.

- **Zvýraznění dramatičnosti příběhu:** Hudba může zvýraznit, podobně jako u filmu, dramatické či naopak méně dramatické momenty v příběhu. Právě především tuto roli má často hudba ve výše zmíněných cut-scénách.
- **Představení charakteru či místa:**[4] Hudba může pomoci dokreslit atmosféru míst, ve kterých se hráč právě nachází. Běžnou praxí v hrách je propojování konkrétních lokalit s určitým soundtrackem nebo souborem soundtracků, které zpracovávají stejný hudební motiv. Například ve hře *The Elder Scrolls V: Skyrim*[5] je odlišná sada skladeb použita vždy ve městech a jiná při procházení se krajinou. Ve hře *The Witcher 3*[6] jsou některé soundtracky spojené s konkrétními místy či postavami. Např. soundtracková stopa *Emhyr var Emreis* se ozve vždy, když se na scéně objeví stejnojmenný charakter, ale také v oblastech s ním spojených. Tato hra pracuje také s leitmotivy, které se objevují napříč soundtrackem, zpravidla v emočně vyjímajících se místech, často při cut-scénách nebo během dialogů. V kontextu hry je lze chápat tak, že odkazují na emotivní, často intimní prožitky hlavního herního charakteru, kterého hráč po valnou většinu hry ovládá.
- **Změna stavu hry:** Hudba často reflektuje nebo předjímá změnu stavu hry, typicky bojovou situaci. To bývá řešeno za pomocí technik horizontální resekvence (např. *The Elder Scrolls V: Skyrim*) a vertikálního remixu (např. *Red Dead Redemption 2*[7]), o kterých bude psáno dále.
- **Zvyšování či snižování dramatického napětí:** Zvyšující se tempo a přidávání dalších hudebních vrstev bývají metody k postupnému zvyšování či uvolňování napětí.
- **Komunikace události hráči:** Často se ve hrách vyskytují také krátké znělky (3-12 vteřin), reagující přímo na určitou situaci ve hře. Těmi může např. splnění úkolu, smrt herního charakteru nebo objevení nové oblasti ve hře.
- **Emoční propojení hráče ke hře:** Výrazné hudební téma pro hru může pomoct dotvořit celkovou náladu celé hry. Takové téma se nemusí ani nacházet během hraní samotné hry, může ale vzbuzovat v hráči další nadšení k hraní hry či ho naladit před samotným hraním – tuto úlohu mají typicky soundtracky znějící hned po spuštění hry, tak jako např. v sérii *The Elder Scrolls*.
- **Hudba jako součást hraní hry:** Specifickou roli hudba zaujímá v případě, kdy je sama součástí hraní hry (gameplaye), tak jako např. v sérii *Guitar Hero*.[8] Tato role je podrobněji rozebrána v kapitole 1.3.3 Hudba jako součást hraní.

[4] Michael Sweet tyto dva body odlišuje. Zde uvedený příklad z *The Witcher 3* však ukazuje, jak se mohou tyto dva body prolnout. Z tohoto důvodu přišlo autoru smysluplné tyto dva body spojit.
[5] *The Elder Scrolls V: Skyrim* [videohra]. 2012. Bethesda Game Studios. Bethesda Softworks.
[6] *The Witcher 3* [videohra]. 2015. CD Projekt. CD Projekt Red.
[7] *Red Dead Redemption 2* [videohra]. 2018. Rockstar Studios. Rockstar Games.
[8] *Guitar Hero* [videohra]. 2005. Harmonix Music System. RedOctane.

1.1.3 Opakování

Zatímco běžná délka filmu se pohybuje mezi 1,5–2,5 hodiny, u hry hráč typicky stráví 10 a více hodin.[9] Rozsáhlé hry pak mohou mít dostatek obsahu na vyšší desítky až stovky hodin, za určitých podmínek (např. u her s generovaným obsahem) lze dokonce považovat potenciál času stráveného u hry za nekonečný. Pokud není hudba v takové hře generativní, což je dnes stále minoritní přístup, tak je pravděpodobné, že se budou jednotlivé stopy soundtracku ve hře opakovat. Aby se předešlo únavě hráče z příliš často opakující se hudby[10], obsahují tyto hry i nižší desítky hodin soundtracku. Například *World of Warcraft*[11] obsahoval k roku 2014 přes 23 hodin hudby.[12] Kromě obrovské délky samotného soundtracku napomáhá variabilitě také proměna orchestrace a přidávání dalších vrstev, tedy vertikální remix, který je popsán dále.

Fakt, že se jednotlivé stopy soundtracku[13] ve hře nutně opakují, má přesah i do skladatelské roviny. Je potřeba počítat s tím, že jednotlivé stopy na sebe musí přirozeně navazovat (předpokládá se, že se opakuje větší počet stop), a to případně i v náhodném pořadí. Je tedy nutné uvažovat o řadě strukturálních parametrů hudby:

- Harmonický plán jednotlivých stop a tonální vztahy mezi stopami.
- Instrumentace a textura. Začátky a konce jednotlivých stop by neměly být příliš kontrastní.
- Tempo. To může sloužit jako spojovací prvek dvou odlišných stop, anebo naopak jako spojovací prvek stop v jiných složkách kontrastnějších.

Potenciálně vysoká četnost opakování také vede k tomu, že by se soundtrack měl vyhýbat výrazných hudebních momentů, které mohou vystoupit do popředí a upoutat hráčovu pozornost. Soundtrack totiž nejlépe plní svou úlohu, pokud hudba působí na podvědomé úrovní.[14] Ve hrách s důrazem na vyprávění mohou být výrazné hudební momenty vyhrazené pro

---

[9] Výjimku pochopitelně představují drobné hry, jejichž dohrání trvá maximálně několik hodin.

[10] V současnosti se stále jedná rozšířený problém v celé řadě her. Jako jeden z důvodů lze vidět fakt, že velkou část dnešního herního trhu (a autorovy zkušenosti) představují nezávislé (tzv. indie) hry, které ve většině případů pracují s omezeným rozpočtem. Kvůli tomu tak nezbývá příliš prostředků na velké množství hudby nebo komplexnější hudební řešení jako je třeba generativní hudba. Komplexnější řešení a (nebo) delší soundtracky mají především velká herní studia u tzn. AAA titulů.
Informace o podílu nezávislých her pochází ze serveru https://vginsights.com/ , který se zabývá analýzou dat o cenách, hodnocení a prodeji her na herní platformě Steam. O zdroji dat, metodologii lze více najít na: https://vginsights.com/about

[11] *World of Warcraft* [videohra]. 2004. Blizzard Entertainment.

[12] SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7. s. 18

[13] Dále také pouze jen jako „stopy".

[14] Norbert J. Schneider, *Handbuch Filmmusik I: Musikdramaturgie im neuen Deutschen Film* (Munich: Verlag Ölschläger, 1990) 2nd Edition. (parafráze dle:
BERNDT, Axel. Adaptive Game Scoring With Ambient Music. In: Monty Adkins – Simon Cummings (eds.). *Music Beyond Airports. Appraising Ambient Music.* Huddersfield: University of Huddersfield, 2019. s. 200. Dostupné z:

výjimečné situace ve hře, jako jsou cut-scény (viz předchozí kapitola). Lze tvrdit, že hudebně výrazný moment v soundtracku jako takovém není sám o sobě problematický – tím se stává až v případě, kdy se příliš často opakuje. V takovém případě utkví v hráčově paměti, což při každém opakování může vést k vyrušení hráče z ponoření se do hry.[15]

1.2 Typy herní hudby

1.2.1 Nediagetická hudba

Nediagetická hudba (jinak také nazývaná extra-diagetická) se v principu neliší od filmu. Pokud herní (či filmové) charaktery hudbu neslyší, ale hráč (v případě filmu divák) ano, jedná se o nediagetickou hudbu. Jde o typ hudby, se kterým se lze v herní produkci nejčastěji. Právě nediagetická hudba má působit především na podvědomé úrovni:

„Běžně se říká, že nejlepší filmové hudby si diváci ani diváci nevšimnou. Výraznější (a neohrabanější) hudba vytrhuje diváka ze zážitku ze sledování filmu tím, že do popředí pro posluchače dávají to, co by mělo být nevědomým prvkem.“[16]

1.2.2 Diagetická hudba

Některé hry využívají i diagetické hudby, tedy takové, která se nachází přímo v herním světě a kterou slyší i herní charaktery. Například v *Red Dead Redemption 2* se v barech nachází nehráčské charaktery[17], které občas hrají na přítomné pianino. O něco komplikovanějším příkladem je klubová hudba v *Cyberpunk 2077*.[18] [19] V této hře se nachází několik klubů, přičemž každý má velmi odlišný charakter. Jedinečnost těchto míst je umocněna právě hudbou tak, že v každém klubu hrají jiné hudební žánry. Současně mají tyto kluby nadstandardně propracovanou simulaci akustiky. Ekvalizace a reverb se dynamicky a velmi realisticky mění, což ostatně platí pro celou hru. Dalším příkladem diagetické hudby, nad kterou má však určitou kontrolu samotný hráč, je rádio. Tento herní prvek je známý především ze série Grand Theft

https://www.researchgate.net/publication/335034399_Adaptive_Game_Scoring_with_Ambient_Music [cit. 30-12-2021])

[15] Jinými slovy imerze. Herní imerze je výrazný fenomén, kterému je věnována celá řada článků. Vede k ní mnoho odlišných cest, ale obecně lze říct, že se jedná o zásadní cíl každé hry. Imerze je totiž to, co hráče u hry drží, a co ho motivuje k tomu, aby hru znovu zapnul. Více o imerzi ve videohrách např. v: BROWN, Emily a Paul CAIRNS. A grounded investigation of game immersion. In: *CHI '04 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2004, 2004-04-24, s. 1297-1300. ISBN 1581137036. Dostupné z: doi:10.1145/985921.986048

[16] SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7. s. 21

[17] Tzv. NPC: non player character.

[18] *Cyberpunk* 2077 [videohra]. 2020. CD Projekt Red. CD Projekt.

[19] *Cyberpunk 2077* využívá ke zpracování zvuku a hudby Wwise, který je dále v této práci podrobněji popsán. Více o zvukovém designu v této hře lze najít v této prezentaci:
WALDER, Collin. *Sounds of Night City: Audio Technology In Cyberpunk 2077* [Záznam z konference]. Game Developers Conference, 20–24.3.2023, San Francisco, CA, USA. Dostupné z: https://www.gdcvault.com/play/1029310/Sounds-of-Night-City-Audio [cit. 18-6-2023]

Auto (např. GTA 5[20]), ale i v *Cyberpunk 2077* se nachází mnoho vozidel, ve kterých si hráč může zapnout rádio a měnit na různé stanice. Každá stanice pak obsahuje odlišnou sadu písniček zpravidla jiného hudebního žánru. Na rozdíl od GTA ale *Cyberpunk* 2077 obsahuje kolem tří hodin originální diagetické hudby, na které se podílelo mnoho umělců a kapel pod pseudonymy hudebníků ze samotné hry. Tato hudba pak zní i v již zmíněných klubech. Hudba přitom často odkazuje na fakta uvnitř světa, ve kterém se samotná hra odehrává. Diagetická hudba má tak v tomto případě bezprecedentní roli v dotváření přesvědčivého světa, ve kterém se hra odehrává – svou délkou, pestrostí, důrazem na detail a provázaností se světem hry.[21]

1.2.3 Hudba jako součást hraní

Zvláštním typem hudby je taková, která je přímou součástí samotné hry – hráč sám hraním hry může hudbu (či nějakou z jejích složek) i generovat. Hudba je tedy interaktivní a přímo reaguje na akce hráče. Na první pohled by se mohlo zdát, že by do této kategorie mohla spadat všechna herní hudba. Zásadní rozdíl ale je v tom, zda hudba reaguje na akce hráče přímo či nepřímo. Většina adaptivních hudebních systémů totiž reaguje nepřímo. Například změna stavu hry při započetí souboje je nepřímou reakcí na akci hráče, a tudíž není ukázkou hudby jako součástí hraní. [22]

Hry, které tímto způsobem pracují s hudbou, bývají často velmi specifické a originální. Z tohoto důvodu jsou zde uvedeny tři odlišné příklady. Tím prvním je hra *Guitar Hero*, ve které se hráč ujímá role rockového kytaristy. Cílem hry je zmáčknout klávesy či ovladač ve správný čas podle vizuálně reprezentovaných „not", čemuž pomáhá také rytmus hudby – písničce, kterou zrovna hráč „hraje", a u které chybí kytarová linka. Ta se v případě správného zmáčknutí generuje identicky k originálu, a pokud hráč chybuje, ozývají se různé „chybné" zvuky.

Druhým případem je hra *Rez.*[23] V ní zní neměnná hudba na pozadí, někdy také synchronizovaná s obrazem. Tu doplňují různé hudební fragmenty, které se k tomuto hudebnímu pozadí synchronizují. Tyto fragmenty spouští hráč tím, že sestřeluje různé geometrické obrazce a jiné objekty ve hře.

Třetím příkladem je hra *Beat Saber* (2018). Jde o rytmickou akční hru, ve které musí hráč ve virtuální realitě strefovat kostky světelnými meči. Pohyb kostek je synchronizovaný s hudbou, a tak hráč dělá pohyby do rytmu hudby. Samotný rytmus hudby je pro orientaci hráče zcela zásadní součást hraní. Při zásahu kostek se ozve pouze jeden zvuk synchronizovaný s hudbou

[20] *Grand Theft Auto 5* [videohra]. 2013. Rockstar North. Rockstar Games.
[21] Nejedná se o zcela nový prvek, např. v *The Elder Scrolls V: Skyrim* bardi v hostincích zpívají v několika variantách píseň, která se odkazuje na historii světa hry. Lze se setkat i s jinými NPC, které si tutéž píseň pobrukují.
[22] SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7. s 24
[23] *Rez* [videohra]. 2001. United Game Artists. Sega.

– v tomto ohledu jsou dva předchozí příklady v oblasti generování hudby výrazně komplexnější.

Ačkoliv je ve všech těchto příkladech řeč o generování hudby, nelze ani v jednom říct, že by hudba byla zcela generativní. Všechny složky hudby jsou totiž předem vytvořené a průběh všech tří her je lineární. Současně je ona „generativnost" přímo vázaná na akci hráče a není tak výsledkem autonomního systému (viz 2. Generativní hudba).

1.3 Technický přístup k herní hudbě

1.3.1 Neadaptivní hudba

Nejjednodušším technickým přístupem k herní hudbě je takový, který žádným způsobem nereaguje na průběh hry. V tomto případě hra obsahuje pouze seznam soundtrackových stop, které hrají v postupném či náhodném pořadí. Tento přístup je častý u strategických her, ve kterých může zdánlivě dramatický stav jako válka trvat i několik hodin (např. *Stellaris*[24]). Jako mezistupeň mezi zcela neadaptivní a adaptivní hudbou lze chápat hudbu v sérii strategií *Civilization* (např. *Civilization V*[25]). V ní se sice stále opakují stejné stopy soundtracku, ty jsou ovšem odlišné podle frakce, kterou si na začátku hry zvolí. Postupem hry se také seznam stop mění tak, jak hráč přechází do modernějších ér – jiná hudba tedy hraje v pravěku a jiná ve renesanci, přičemž jde stále o variace na stejné téma, které je specifické pro danou frakci.

1.3.2 Interaktivní a adaptivní hudba

Obě kategorie interaktivní a adaptivní hudby (či obecně zvuku) lze shrnout pod pojmem *dynamická hudba* (zvuk).[26] *Adaptivní hudba* v reálném čase reaguje na akce hráče pouze nepřímo – reaguje na proměnu stavu hry. Změnou stavu hry může být třeba přesun hráčského charakteru do lokace ve hře s jinou hudbou nebo soubojová situace. Stav hry přitom nemusí být vůbec v rukou hráče, např. soubojová situace může nastat i zcela náhodně, nezávisle na jeho akci.

Interaktivní hudba se od adaptivní hudby odlišuje tím, že reaguje na akci hráče přímo. Příklady lze najít v raných videohrách, jako např. v *Super Mario Bros*[27]. V té se po sebrání předmětů ve hře ozvou krátké melodické znělky. Tyto znělky také nahrazují neexistující zvukové efekty z důvodu omezených technických možností tehdejší doby. Někdy byly tyto znělky také

[24] *Stellaris* [videohra]. 2016. Paradox Development Studio. Paradox Interactive.
[25] *Civilization V* [videohra]. 2010. Firaxis Games. 2K, Aspyr.
[26] COLLINS, Karen. Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design. Cambridge, Mass: MIT Press, 2008. ISBN 978-0-262-03378-7. s. 139
[27] *Super Mario Bros* [videohra].1985. Nintento.

vytvářeny tak, aby zapadaly do hudby znějící na pozadí.[28] Novějšími příklady interaktivní hudby pak může být dříve zmíněný *Rez* nebo *Guitar Hero*.

Cílem této podkapitoly není vyčerpávajícím způsobem popsat jednotlivé techniky, se kterými se lze v herní hudbě setkat. Jde pouze o představení základních konceptů tak, aby byl srozumitelný rozdíl mezi dynamickou a generativní hudbou, které se věnují poslední dvě kapitoly této práce.

Jelikož jsou horizontální resekvencování a vertikální remix[29] v současnosti nejběžnějšími technikami adaptivní hudby ve hrách, jejich konkrétní ukázky realizované ve FMOD[30] lze najít v kapitole 1.4 Zvukové enginy.

1.3.2.1 Horizontální resekvencování

Horizontální resekvencování v té nejjednodušší podobě znamená, že při změně stavu přeplyne (přes tzv. crossfade) jedna stopa soundtracku do nové, která odpovídá aktuálnímu stavu hry. Pokročilejší systém byl představen v *Monkey Island 2: KeChuck's Revenge*.[31] Ten umožňuje počkat na skončení hudební fráze předtím, než se přepne na novou stopu (či na jiný čas téže stopy), a vytvořit tak mezi nimi plynulý přechod. Stejné možnosti nabízí v současnosti i moderní herní zvukové enginy (tzv. audio middleware) jako je Wwise[32] nebo FMOD. Tyto přechody často doprovází krátká znělka (tzv. stinger). Ta funguje jako krátký přechod mezi odlišnou hudbou, která odpovídá odlišným stavům hry. Může také pomoci zakrýt zvukové, popř. i hudební nedokonalosti v přechodu.[33]

1.3.2.2 Vertikální remix

Vertikální remix mění hlasitost jednotlivých vrstev v hudbě. Jedna stopa se může skládat z několika různých vrstev (např. perkuse, smyčce na pozadí a melodie), které lze individuálně zesilovat či zeslabovat na základě proměny stavu hry. To může být výhodné řešení v případě, že se stav hry často proměňuje, a pouhý horizontální remix by tak (např. i se stále se opakující znělkou – stingerem) mohl začít působit rušivě.[34] Často se ale ve hrách vyskytují oba postupy společně, kdy je typicky horizontální resekvencování vyhrazeno pro jasné změny stavu hry jako

---

[28] BERNDT, Axel. HARTMANN, Knut. The Functions of Music in Interactive Media. In: Spierling, U., Szilas, N. (eds) *Interactive Storytelling.* ICIDS 2008. Lecture Notes in Computer Science, vol 5334. Springer, Berlin, Heidelberg. 978-3-540-89454-4. s. 4. Dostupné z: https://doi.org/10.1007/978-3-540-89454-4_19 [cit. 20-6-2023]

[29] Pojmy jsou převzaty z *Writing Interactive Music For Video Games* od Michaela Sweeta, lze se ale setkat i s pojmy horizontální remix nebo vertikální resekvencování, přičemž je myšleno to samé.

[30] *FMOD* [software]. Firelight Technologies. Dostupné z: https://www.fmod.com/

[31] *Monkey Island 2: KeChuck's Revenge* [videohra]. 1991. LucasArts

[32] *Wwise* [software]. Audiokinetic. Dostupné z: https://www.audiokinetic.com/en/products/wwise/

[33] SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7. s. 45, 143–152.

[34] Tamtéž, s. 46

průzkum/boj nebo přesun do jiné lokace, a vertikální remix ke stupňování dramatické situace, např. souboj různé intenzity. Například v případě hry *Cyberpunk 2077* přechází hudba na odlišnou stopu při potenciálním nebezpečí, a při započatí samotného boje se pak přidávají další vrstvy.

1.3.2.3 Tempová manipulace

Dynamická změna tempa je v herním soundtracku poměrná vzácná, jelikož jde o postup poměrně náročný na CPU. Teoreticky možným řešením může být zpomalování již předpřipravených stop. Toto řešení ale většinou není zvukově kvalitní, resp. kvalitnější řešení jsou pak opět náročná na výkon. Příklady, které změny tempa využívají, pak syntetizují hudbu v reálném čase za pomocí MIDI a virtuálních nástrojů. Staré hry jako *Space Invaders*[35], *Asteroids*[36], nebo *Cervii*[37] využívaly omezených výpočetních a zvukových prostředků své doby pro zvyšování napětí pomocí postupně se zvyšujícího tempa. Jelikož byl zvuk v reálném čase syntetizován, nebyl problém s tempem manipulovat. Kvalita a možnosti těchto relativně primitivních virtuálních nástrojů byly ve srovnání s dnešní dobou nízké, přesto se však tento zvuk stal ikonickým. I díky tomu se stále setkáváme s měnícím se tempem i v novějších hrách – hráči jsou stále ochotni přijmout tuto „lo-fi" estetiku, která umožňuje bez problémů syntetizovat zvuk v reálném čase. Příkladem může být *Super Mario Galaxy*[38], kde hráč ovládá kutálící se míč. Čím rychleji se míč pohybuje, tím vyšší tempo má doprovodná hudba. A i když jsou zde syntetické zvuky o něco propracovanější než u starých her, odkaz na estetiku jednoduchých syntezátorů ve hrách z přelomu 70. a 80. let je zcela zřejmý.[39]

1.3.2.4 Efekty

Efekty, jinak také DSP (digital signal processing), jsou ve hrách poměrně často využívány. Diagetická huba může být významně efektována filtry a reverbem při simulaci akustického prostoru, případně i jinými efekty proto, aby odpovídaly zdroji, ze kterého mají vycházet (např. distortion pro rádio). Efektovaná ale může být i nediagetická hudba: například v *Call of Duty*[40] i jiných bojových hrách se často používá low-pass filtr na všechen zvuk (tedy i včetně hudby) v případě, kdy se zdraví hráčova charakteru blíží nule. To slouží také jako velmi efektivní upozornění hráče na důležitou situaci ve hře.[41]

---

[35] *Space Invaders* [videohra]. 1978. Taito.
[36] *Asteroids* [videohra]. 1979. Atari, Inc.
[37] *Cervii* [videohra]. 1993. Vladimír Chvátil.
[38] *Super Mario Galaxy* [videohra]. 2007. Nintendo.
[39] SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7. s. 47
[40] *Call of Duty* [videohra]. 2003. Infinity Ward. Activision.
[41] Tamtéž, s. 47

1.3.2.5 Proměna instrumentace a aranžmá

Tato technika je velmi podobná vertikálnímu remixu. Příkladem může být *Monkey Island 2: LeChuck's Revenge*, kde hráč přechází mezi odlišnými prostory. Místo toho, aby se změnila hudební stopa, změní se nástroj, kterým je všudy přítomná melodie hrána. Tuto techniku lze nalézt už v docích na samotném začátku hry. Na rozdíl od vertikálního remixu je ale proměna instrumentace typicky používána v soundtracku pracujícím s MIDI a virtuálními nástroji, tedy se syntézou v reálném čase. Stejného efektu lze dosáhnout i pomocí vertikálního remixu, ten ale vyžaduje více operační paměti. Náročnost na paměť pak dále roste spolu s komplikovaností systému a množstvím variant.[42]

1.3.2.6 Tónová manipulace

Tónová manipulace je ve hrách spíše neobvyklá, přičemž komplikovanější řešení by již bylo lepší považovat za generativní hudbu.[43] Příklad spadající stále do adaptivní hudby může být změna frekvence celých pasáží přes tzv. pitch-shift, byť se technicky vzato jedná spíše o efekt. Historicky vzato se jednoduché transpozice celých pasáží používaly jako řešení pro omezené množství paměti na raných zařízeních. Jednoduché transpozice sekvencí lze najít např. v arkádové hře *Rally X*.[44] Karen Collins[45] uvádí jako příklad tónové manipulace i novější hru *Legend of Zelda: Twilight Princess*. V té by se měla v závěrečné bojové scéně zvyšovat výška hudby při každém úspěšném zásahu nepřítele. Při ověřování tohoto příkladu na základě videí se však nezdá, že by to byla pravda – ono zvyšování výšky se zdá předem určené, a jen někdy shodou okolností dochází k synchronizaci se samotným hraním.

Michael Sweet[46] uvádí jako příklady tónové manipulace také hry, u kterých je hudba součástí hraní: *Rez* a *Child of Eden*.[47] Není ale zřejmé, co má konkrétně na mysli. Při srovnání dvou videí z hraní *Rez* se zdá, že sestřelované objekty mají vždy stejnou tónovou výšku, a rozdíl závisí pouze na pořadí, v jakém je hráč sestřelí. V případě sestřelení více cílů naráz se pak může ozvat dodatečný zvuk. Sweet dále uvádí jako příklady *Asheron's Call II: Fallen Kings*[48] a *No One Lives Forever*[49], ve kterých by měl být možný adaptivní přechod mezi odlišnými sledy akordů. To se však nepodařilo ověřit.

---

[42] Tamtéž, s. 49, 221
[43] Tamtéž, s. 49
[44] COLLINS, Karen. Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design. Cambridge, Mass: MIT Press, 2008. ISBN 978-0-262-03378-7. s. 148
[45] Tamtéž.
[46] SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7. s. 49
[47] *Child of Eden* [videohra]. 2011. Q Entertainment. Ubisoft.
[48] *Asheron's Call II: Fallen Kings* [videohra]. 2002. Turbine Entertainment Software. Microsoft Game Studios.
[49] *No One Lives Forever* [videohra]. 2000. Monolith Production. Fox Interactive.

1.3.3 Generativní hudba

Svou vlastní kategorií je generování hudby ke hře v reálném čase. Generování hudby může být také adaptivní. Na rozdíl od vykomponované hudby může generování zaručit vždy unikátní soundtrack, jelikož je jeho délka prakticky nekonečná. Ve hrách se však generativní hudba příliš nevyskytuje, protože její implementace je podstatně komplikovanější než u adaptivní hudby. Jelikož se tato práce zabývá detailně konkrétním experimentálním řešením generování hudby pro hry, je generativní hudbě věnována samostatná kapitola – více o generativní hudbě ve 2. Generativní hudba ve hrách.

1.4 Zvukové enginy (implementace hudby do hry)

1.4.1 O zvukových enginech obecně

Jak již bylo zmíněno, hry často využívají k implementaci zvuku (a tedy i hudby) tzv. audio middleware. Obecně pojem middleware v softwarovém inženýrství znamená softwarovou komponentu, která slouží jako prostředník mezi různými částmi systému nebo aplikace, zpracovává požadavky a odpovědi a poskytuje různé funkcionality a operace mezi nimi. V herním vývojářství bývají middlewary typicky používány jako specializované enginy či vývojářské sady (např. se zaměřením na zvuk, fyziku nebo třeba zobrazování oblohy), které jsou propojeny s herním enginem. Middlewary někdy mají také vlastní uživatelské rozhraní, které usnadňuje designérům pracovat na hře, jelikož se nemusí zabývat psaním kódu.[50]

Herní enginy (Unity[51], Unreal Engine[52]) mají často své vlastní zvukové systémy. Ty mohou v jednoduchých případech stačit, komplikovanější řešení ale (např. adaptivní hudba nebo dynamická ekvalizace) vyžaduje vlastní programování. Z tohoto důvodu může být efektivnější sáhnout po již hotových řešeních v podobě audio middleware. Uživatelské rozhraní u takového zvukového enginu navíc usnadňuje práci zvukovým designerům a hudebním skladatelům. Ti tak můžou pracovat bez nutnosti vlastního programování a zkoušet, jak jednotlivé zvukové prvky fungují v kontextu hry – různé stavy hry je možné v takových softwarech simulovat. Audio middleware vše uchovává do zdrojových souborů, které lze efektivně integrovat do hry.[53] Audio middleware může také usnadnit vývoj hry pro vícero herních platforem.[54]

[50] Tamtéž, s. 247
[51] *Unity* [software]. Unity Technologies. Dostupné z: https://unity.com/
[52] *Unreal Engine* [software]. Epic Games. Dostupné z: https://www.unrealengine.com/
[53] HERBER, Norbert. *Amergent Music: behavior and becoming in technoetic & media arts*. Disertační práce. Plymouth: University of Plymouth, 2010. s. 17. Dostupné z: https://pearl.plymouth.ac.uk/handle/10026.1/307 [cit. 27. 12. 2021]
[54] SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7. s. 248

V současnosti existuje řada různých zvukových enginů, přičemž mezi nejčastěji využívané patří FMOD a Wwise, a dále pak například Fabric[55], Allegro library[56] nebo OpenAL.[57] Poslední dva příklady ovšem neobsahují uživatelské rozhraní a jedná se pouze o knihovny s vlastními API (Application Programming Interface), která umožňuje vývojářům interakci mezi knihovnami a jejich aplikací; nenabízejí tedy snadnější přístup pro zvukové designery nebo skladatele, ale pouze rozšiřují možnosti zpracování zvuku. Dále existují řešení, která patří konkrétním herním studiím a nejsou veřejně přístupná, např. RAGE[58] využívá svůj vlastní zvukový systém.[59]

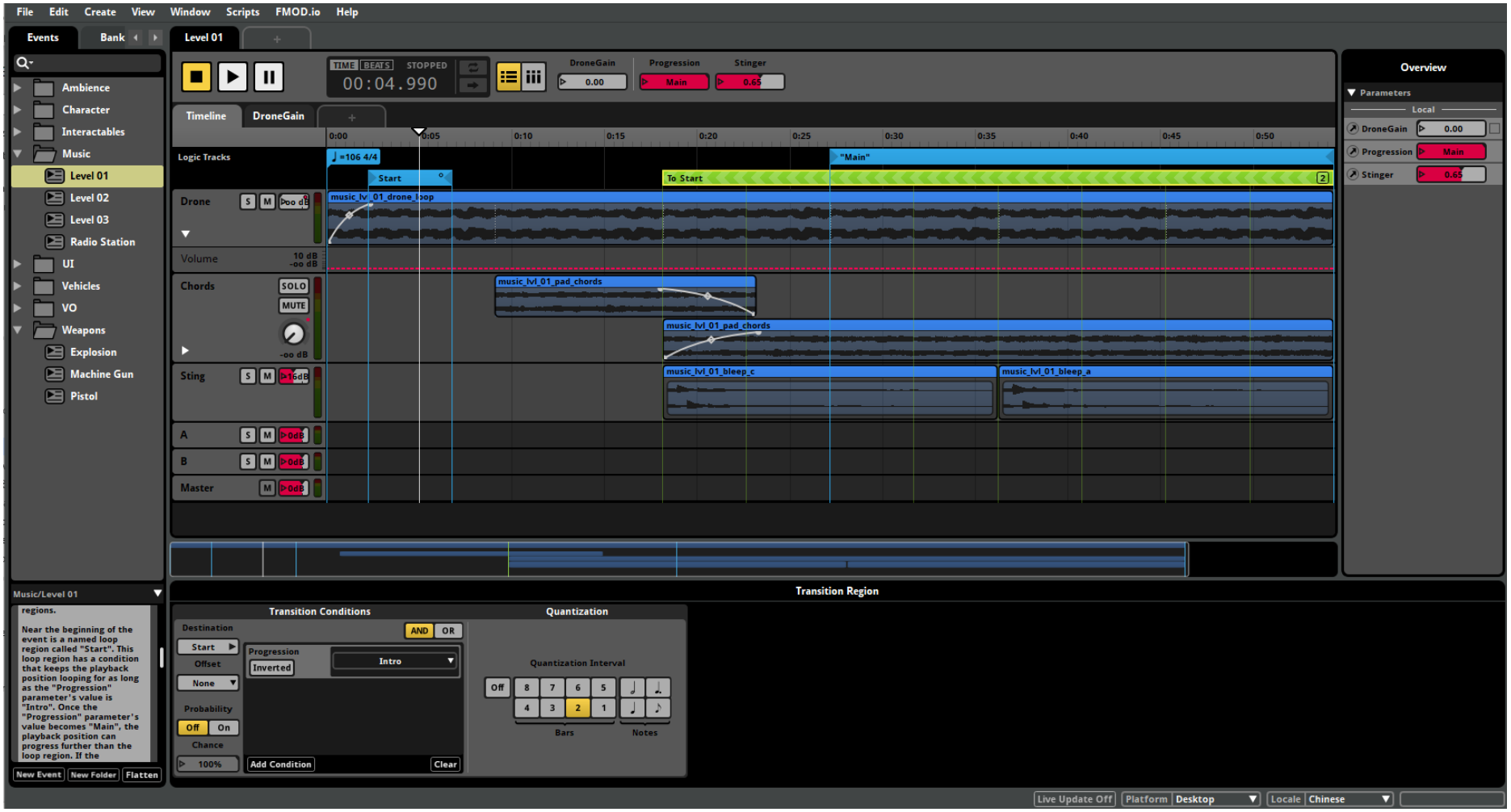
1.4.2 Ukázka implementace hudby ve zvukovém enginu: FMOD

Tato ukázka má posloužit k získání lepší představy o tom, jaké možnosti moderní audio middleware nabízí. Současně je na tomto příkladu možné ukázat konkrétní realizaci horizontální resekvence a vertikálního mixu. FMOD jako zvukový engine byl pro tento příklad vybrán z toho důvodu, že se jeho rozhraní ve srovnání s Wwise více podobá DAW (Digital Audio Workstation) jako je ProTools nebo Reaper. Z tohoto důvodu by mohly být ukázky také lépe srozumitelné. FMOD je poměrně rozsáhlý software s mnoha možnostmi – příklady zde uvedené zdaleka nevyčerpávají všechny možnosti, které FMOD nabízí.[60] I když jsou ukázky zaměřené na konkrétní zvukový engine, obecně lze říct, že většina ostatních audio middleware nabízí podobné možnosti.

Při práci s FMOD je třeba rozlišit dvě činnosti: implementaci v herním enginu, která může zahrnovat i vlastní programování, a design, který probíhá v samostatné aplikaci FMOD Studio. Veškerý zvuk (tedy i hudba) se ve FMOD řídí přes události (eventy). Ty se spouští za různých podmínek ve hře. V případě herního enginu Unity je možné využít volně dostupné, předem připravené skripty, které jsou součástí balíčku FMOD Studio Unity Integration. Je také možné napsat nové skripty s použitím FMOD API. Z herního enginu tedy plynou data o tom, jaká událost se má spustit, a případně s jakými parametry. Ve FMOD Studio se jednotlivé události a parametry nastavují. Po dokončení nastavení se z FMOD Studio vyexportují soubory, které se následně nahrají do rozšíření herního enginu, které je součástí integračního balíčku.

---

[55] *Fabric* [software]. Tazman-Audio. Dostupné z: https://www.tazman-audio.co.uk/fabric20
[56] *Allegro 5* [software]. Open source. Dostupné z: https://liballeg.org/
[57] *OpenAL* [software]. Creative Technology. Dostupné z: https://www.openal.org/
[58] Rockstar Advanced Game Engine. V tomto neveřejném enginu jsou vytvořené hry studia Rockstar, např. GTA 5 nebo Red Dead Redemption 2.
[59] MACGREGOR, Alastair. *The sounds of GTA5: Rage Audio Features* [záznam z konference]. Game Developers Conference, 17–21.3.2014, San Francisco, CA, USA. Dostupné z: https://www.youtube.com/watch?v=EUN1j-3IPW0 [cit. 18.6.2023]
[60] Více o funkcích FMOD Studia: https://www.fmod.com/docs/2.02/studio/fmod-studio-concepts.html [cit. 18.6.2023]

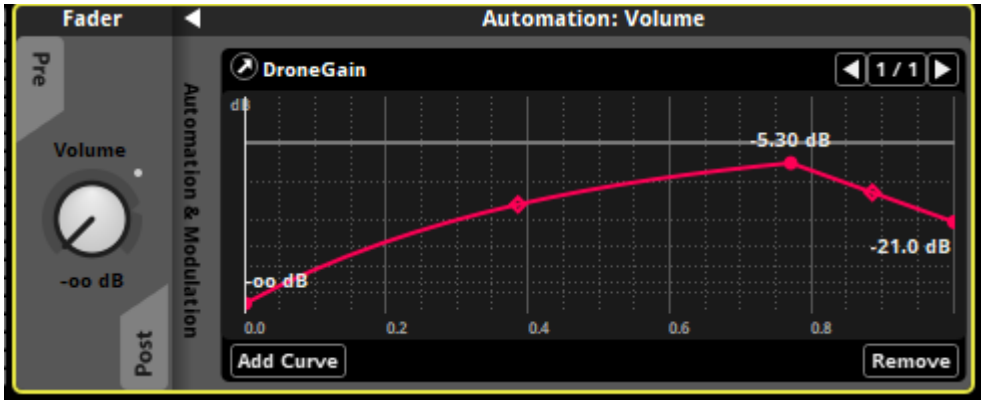

File Edit Create View Window Scripts FMOD.io Help
Events
Bank
Level 01
Ambience
Character
Interactables
Music
Level 01
Level 02
Level 03
Radio Station
UI
Vehicles
VO
Weapons
Explosion
Machine Gun
Pistol
TIME BEATS STOPPED
00:04.990
DroneGain
0.00
Progression
Main
Stinger
0.65
Overview
Parameters
Local
Timeline
DroneGain
Logic Tracks
=106 4/4
"Main"
Start
To Start
Drone
music_lvl_01_drone_loop
Volume
Chords
SOLO
MUTE
music_lvl_01_pad_chords
Sting
music_lvl_01_bleep_c
music_lvl_01_bleep_a
A
B
Master
Music/Level 01
regions.
Near the beginning of the event is a named loop region called "Start". This loop region has a condition that keeps the playback position looping for as long as the "Progression" parameter's value is "Intro". Once the "Progression" parameter's value becomes "Main", the playback position can progress further than the loop region. If the
New Event
New Folder
Flatten
Transition Region
Transition Conditions
Destination
Start
AND
OR
Offset
None
Progression
Inverted
Intro
Probability
Off
On
Chance
100%
Add Condition
Clear
Quantization
Quantization Interval
Off
8
7
6
5
4
3
2
1
Bars
Notes
Live Update Off
Platform
Desktop
Locale
Chinese

Obrázek č. 1

1.4.2.1 Ukázka: Horizontální resekvencování

Na obrázku č. 1 je vidět jednoduchá ukázka horizontálního resekvencování.[61] Adresářová struktura pod záložkou „Events“ v levém sloupci funguje i jako název události na straně herního enginu. Ve středu horní části obrazovky je nastavitelná logika jednotlivých stop eventu. Po pravé straně jsou uvedené všechny parametry dostupné pro danou událost – v tomto případě Music/Level 01. Parametr „Progression“ má dva stavy: „Intro“ a „Main“. Pokud je hra ve stavu „Intro“, stopa se opakuje ve smyčce podle logické stopy pojmenované „Start“. V případě, že se stav hry náhle přepne na „Main“, událost pokračuje plynule dále na časové ose, a místo toho začne později opakovat ve smyčce jednotlivé zvukové stopy podle logické stopy „Main“. Další, zeleně vyznačená logická stopa „To Start“ je pak kvantizována – její nastavení je vidět na stejném obrázku dole. Momenty kvantizace jsou na časové ose zobrazeny jako zelené vertikální čáry. Logická stopa „To Start“ způsobuje, že pokud se stav hry znovu změní na „Intro“, událost na časové ose pokračuje až do následující zelené čáry, a poté začne znovu přehrávat smyčku „Start“.

Další parametr „DroneGain“ pak ovládá automatizaci hlasitosti stopy „Drone“. Hodnota parametru je reálné číslo o minimální hodnotě 0 a maximální hodnotě 1. Tento parametr je namapovaný na automatizační křivku, která je zobrazena na obrázku č 2. Takto lze ve FMOD ovládat mnoho různých efektů.



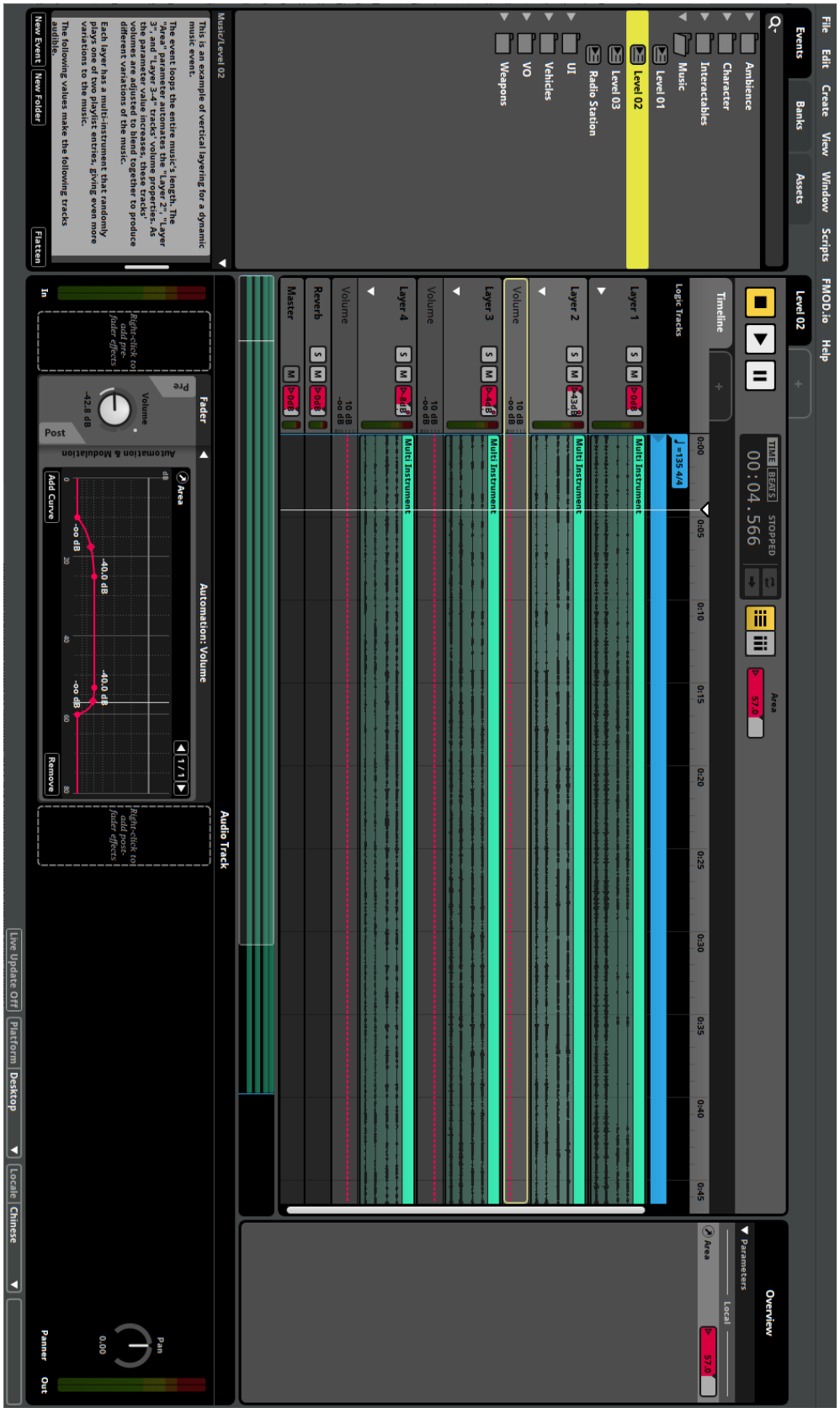
Obrázek č. 2

Poslední zmínění hodnou částí této ukázky horizontální resekvence je implementace stingeru. Ten je navázaný na poslední parametr „Stringer“. Pokud tento parametr dosáhne hodnoty jedna, spustí se „nástroj“[62] ze zvukové stopy „Sting“. Samotný nástroj má pak nastavenou kvantizaci, reaguje tedy na tempo celého eventu. Podle aktuální pozice na časové ose pak

[61] Použité ukázky z FMOD pochází z ukázkového „výukového“ projektu, které je součástí instalace programu. Tento projekt je místy upravený – autor experimentoval s FMOD právě v tomto projektu za účelem zjistit více o fungování programu.

[62] Ve FMOD je to takto nazvané – Single/Multi Instrument. Tento „nástroj“ obsahuje zvukový soubor, ale také další nastavení, mimo jiné i to, jak se má v dané logické stopě nástroj chovat.

spustí buď první nástroj „music_lvl_01_bleep_c“ anebo „music_lvl_01_bleep_a“. Přehrává se přitom vždy od začátku, a nikoliv od aktuální pozice na časové ose.



Obrázek č. 3

1.4.2.2 Ukázka: Vertikální remix

Na obrázku č. 3 je ukázka vertikálního remixu ve FMOD, který využívá tzv. multinástroje (multiinstruments). Všechny multinástroje mají přiřazené dva nástroje, každý s odlišnou nahrávkou. Ty jsou vybírány náhodně, přičemž je možné nastavit jejich pravděpodobnost zvolení (obrázek č. 4). Toto slouží pro zvýšení variability hudby. Samotný vertikální remix jakožto adaptivní technika se řídí jedním parametrem „Area“. Tento parametr ovládá automatizace hlasitostí všech těchto multinástrojů tak, aby se přirozeně prolnuly. V tomto příkladu stále zní stopa „Layer 1“, vyšší vrstvy se postupně zesilují a zeslabují při zvyšování hodnoty parametru. Na obrázku č. 4 dole je vidět automatizační křivka pro „Layer 2“.

Obrázek č. 4

1.4.2.3 Shrnutí

Po uvedených ukázkách by mělo být srozumitelnější, jaké jsou dnes běžně dostupné možnosti pro tvorbu adaptivní herní hudby. Jde pouze o jednoduché ukázky, zaměřené spíše na funkce užitečné při tvorbě adaptivní hudby. Nejde tedy o vyčerpávající popis všech funkcí FMOD. Jak již bylo zmíněno, FMOD obsahuje řadu efektů, které je možné přes parametry ovládat, ale také dynamický mix. Dostupná je také řada externích pluginů, které funkcionalitu programu dále rozšiřují. Současně FMOD obsahuje také API, která umožňuje naprogramovat funkce v základním programu nedostupné.

2. Generativní hudba ve hrách

2.1 Stručná historie a definice generativní hudby

Generativní hudba se v mnoha případech překrývá s algoritmickou hudbou, tedy takovou, která je vytvořena za pomocí algoritmů. Ačkoliv se o algoritmech mluví především ve vztahu k programování, jde o obecný pojem: algoritmus je přesný návod či postup k vyřešení dané úlohy. Za algoritmickou skladbu tak lze považovat třeba *Music of Changes* (1951) od Johna Cage, kterou skladatel napsal bez použití počítače. John Cage si předem stanovil strukturu celého díla a sadu pravidel, podle kterých se má kompozice rozvíjet.[63] [64] Podstatnou součástí díla byl přitom indeterminismus – to Cage vyřešil tak, že si místo vlastního rozhodnutí hodil mincí. Tento příklad ukazuje, že algoritmická hudba a indeterminismus (náhoda) [65] nejsou vzájemně vylučující, ale naopak se často vzájemně prolínají. Algoritmická hudba ale může při předem stanovených vstupních hodnotách (tedy bez použití náhodných hodnot během procesu) zcela deterministická, resp. jakákoliv přísně deterministická kompozice naplňuje definici algoritmické hudby.

Co generativní hudbu od algoritmické hudby odlišuje, je autonomnost systému, ze kterého hudba vzniká. Takový systém nutně nemusí být sadou přísných pravidel v podobě algoritmu, ale může jít o jakoukoliv obecnější vlastnost. Hudba generovaná větrem na větrnou zvonkohru je jednoduchým příkladem nealgoritmické generativní hudby: autor[66] rozhodne o nástroji (např. o typu zvonkohry) a o tom, kdy a na jakém místě má být použit. To ovlivňuje jednak prostorové vnímání instalace, ale případně také to, jak moc vítr může interagovat se zvonkohrou. Autor má tedy rozhodující roli v řadě otázek, které však nejsou přísnými postupy tak, aby naplňovaly definici algoritmu. Stále se přitom jedná o autonomní systém, jelikož zvuk přímo závisí na

[63] PRITCHETT, James. The Music of John Cage. Cambridge University Press, 1993. ISBN 0-521-56544-8. s. 78–83.

[64]NATTIEZ, Jean-Jacques. SAMUELS, Robert. *The Boulez–Cage Correspondence*. Cambridge University Press, 1995. ISBN 0521485584. s. 105–107

[65] V případě užití počítače jde ve skutečnosti o pseudonáhodu počítače (resp. algoritmy pro generování náhodných čísel) jsou totiž svou podstatou deterministické. Na rozdíl od hardwarových generátorů náhodných čísel, které využívají např. kvantových jevů pro získávání náhodných hodnot, počítače produkují pseudonáhodná čísla. K tomu se používá jako vstupní hodnota tzv. seedu, který lze buď pevně stanovit (a tak se vždy dostat ke stejnému výsledku), anebo ho stále měnit: nejběžnější programovací jazyky typicky používají jako seed hodnotu systémového času. Toto je běžná praxe: ve skutečnosti existují i indeterministické algoritmy nebo takové, které mohou vracet chybné výsledky (např. z důvodu tzv. race-conditioningu).

[66] Abychom se nemuseli zabývat debatou, zda se jedná o hudbu či umělecké dílo, představme si, že zvonkohra je součástí umělecké instalace.

větru, který je prakticky indeterministický a autor díla na něj nemá další vliv. Vítr a zvonkohra spolu utvářejí jeden systém, který ovšem není abstraktní, ale fyzický.[67]

Ačkoliv se algoritmická a generativní hudba tímto způsobem principiálně liší, pojmem generativní hudba se ve skutečnosti často odkazuje na díla, která naplňují obě tyto kategorie. I z tohoto důvodu se často mluví o generativní hudbě také jako o hudbě využívající algoritmy, která je ale na rozdíl od algoritmické hudby produkována v reálném čase.[68]

Se samotným pojmem generativní hudby přišel skladatel Brian Eno, ačkoliv se nejedná o zcela novou myšlenku: „*The idea of generative music is not original to me (though I think the name is). There have been many experiments towards it over the years, and indeed a lot of my interest was directly inspired by Steve Reich's sixteen tape pieces such as Come Out and It's Gonna Rain.*"[69]

Brian Eno je jednou z nejvýraznějších osob na poli generativní hudby, podstatnou roli hrál i v uplatnění generativní hudby ve videohrách (viz dále). Jedním z jeho dlouhodobých zájmů bylo vytváření „strojů" a „systémů", které by vytvářely hudbu z materiálu a procesů jím specifikovaných, ale v ne předem daných kombinacích a interakcích. To zprvu realizoval dvěma (a později i více) melodickými cykly o různých délkách, které se překrývají a mají velmi vzdálené body synchronizace.[70] *Music for Airports, On Land* nebo *Thursday Afternoon* jsou příklady využívající v různých variacích tyto „automatické" systémy. V audiovizuálních instalacích pak používal nahrávky z kazet, které byly v některých případech zorganizovány tak, že jejich synchronizační bod byl až 14 let. S nástupem počítačů pak Brian Eno vnímal potenciál skutečně realizovat svou představu generativní hudby, nicméně z důvodu neznalosti programování si do žádného konkrétního díla nepustil. Nezávisle na Brianovi Eno ale v roce 1994 vyšel program Koan[71], software pro generování hudby. Některé skladby na něm byly přitom jasně inspirované Enovou hudbou. Ne dlouho poté pak Brian Eno začal sám tento program používat k vlastní tvorbě.[72] Právě instalací *Generative Music 1* z roku 1996, vytvořené pomocí Koanu, pak defacto vytvořil a ustanovil pojem generativní hudby.[73]

---

[67] Rozdíl mezi fyzickými a abstraktními procesy je podrobněji popsán v:
DORIN, Alan. Generative processes and the electronic arts. In: *Organised Sound* [online]. 2001, s. 47-53 [cit. 2023-06-30]. ISSN 1355-7718. Dostupné z: doi:10.1017/S1355771801001078

[68] COLLINS, Nick. The Analysis of Generative Music Programs. *Organised Sound* [online]. 2008, **13**(3), 237-248 [cit. 2023-06-30]. ISSN 1355-7718. s. 238. Dostupné z: doi:10.1017/S1355771808000332

[69] ENO, Brian. Faber, London, UK, 1996. ISBN 0-571-17995-9. s. 332

[70] Synchronizačním bodem je zde myšlen moment, od kterého se hudba začíná doslovně opakovat.

[71] *SSEYO Koan Interactive Audio Platform* [software]. 1994. SSEYO.

[72] ENO, Brian. A Year With Swollen Appendices. Faber, London, UK, 1996. ISBN 0-571-17995-9. s. 330–332

[73] COLLINS, Nick. The Analysis of Generative Music Programs. *Organised Sound* [online]. 2008, **13**(3), 237-248 [cit. 2023-06-30]. ISSN 1355-7718. s. 239. Dostupné z: doi:10.1017/S1355771808000332

2.2 Užití generativní hudby ve hrách

Užití generativní hudby ve hrách není příliš časté, a to pravděpodobně z vícero důvodů: [74]

- Generativní hudba v rozumné zvukové kvalitě má podstatně vyšší nároky na výpočetní výkon, což v důsledku omezuje jiné složky hry, jako jsou grafické detaily nebo komplexnější fyzika.
- Generativní hudba je často špatně předvídatelná a je komplikované jí ovládat. Vygenerovaná hudba může být méně zajímavá než hudba vykomponovaná. Hrozí také, že i když bude hudba stále jiná, nakonec bude působit vlastně stále stejně a monotónně.
- Vývoj systému pro generativní hudbu a následná tvorba v něm klade vyšší nároky na vývojáře a skladatele. Dává tedy smysl, že místo investic do generativní hudby se herní studia spokojí se současnými řešeními, obzvlášť má-li generativní hudba ve hrách negativa zmíněná v předchozích dvou bodech.

Výzkum na poli generativní hudby ve hrách je v poměrně rané fázi, a ani terminologie není ustálená. Nejsou také jasně vymezené hranice toho, co už je generativní a co stále (pouze) adaptivní hudba.[75] V odborné diskusi není například shoda na tom, zda pouhá variabilita, která je důsledkem interakce hráče se hrou (např. v *Rez*), již představuje generativní hudbu. Podle kritérií uvedených v předchozí kapitole ale nikoliv, jelikož se nejedná o autonomní systém. Konkrétní hudební prvky se ozývají na základě lidského vstupu, byť jsou např. v případě *Rez* algoritmy kvantizované tak, aby zapadaly do hudby na pozadí.

Jiný přístup k rozdělení toho, co je a co není generativní hudba ve hrách zvolil Karen Collins. Ten využil terminologii z článku Rene Woolera a dalších. V něm Wooler rozlišuje mezi třemi typy algoritmů: analytickým (ten je dále irelevantní), transformačním a generativním:

"Transformational algorithms tend not have a significant impact on the general musical predisposition of the data representations or the actual size of the data, but can alter the information. For example, an algorithm that transposes individual notes retains the parameters and structural relations of note collection as representation, but alters the pitch value of each note. A retrograde algorithm that reverses the order of a phrase retains the sequential note representation while transforming the structural pattern; but, in the end, the general musical predisposition of the transformed phrase is unaltered.

[74] PLUT, Cale a Philippe PASQUIER. Generative music in video games: State of the art, challenges, and prospects. *Entertainment Computing* [online]. 2020, **33** [cit. 2023-06-30]. ISSN 18759521. Dostupné z: doi:10.1016/j.entcom.2019.100337
Tento článek uvádí řadu příkladů generativní hudby ve hrách a navrhuje vlastní taxonomii pro generativní hudbu ve hrách. Některé zde uvedené příklady jsou však sporné.
[75] Tamtéž, s. 2.

Generative in the context of this framework means "musically generative" and is defined as follows. An algorithm tends toward being generative when the resulting data representation has more general musical predisposition than the input and the actual size of the data is increased. For example, a chaos music algorithm that takes a number as a seed and returns a sequence of notes is generative."[76]

V případě uplatnění transformačního algoritmu v hudbě lze také mluvit o rekombinační hudbě. Příkladem rekombinační hudby může být otevřená forma (např. Stockhausenův *Klavierstück XI*). Rozdíl je však v tom, že při provedení není výsledná forma výsledkem algoritmu, ale vědomého rozhodnutí účinkujícího. Ve hrách ale žádný takový účinkující není, a tak toto musí být řízeno algoritmem, který může případně pracovat také s daty ze hry, tedy nějakým způsobem reagovat na akce hráče.[77] Adaptivní hudba tedy primárně využívá algoritmů, které jsou transformační, nikoliv generativní.
Následující příklady využívají do určité míry právě generativních algoritmů.

2.2.1 Ballblazer (1984)

Ballblazer[78] je obecně považován za první hru, ve které je využit generativní soundtrack. Hlavní téma *Song of the Grid* je ve hře generováno technikou „riffology", vyvinutou programátorem a skladatelem Peterem Langstonem. Melodie se staví z výběru krátkých melodických segmentů (rifů), které se náhodně proměňují, ovšem v rámci předem stanovených pravidel. Jednotlivé melodické segmenty se variují vynecháváním not a proměnou tempa. Vedle melodie je generován (již z menšího výběru) také doprovod. I když se v podstatě vlastně jedná o randomizované horizontální resekvencování a vertikální remix, lze již mluvit o generativní hudbě – rify mají blíže spíš jednotlivým notám než uceleným hudebním myšlenkám. Ty jsou sestavovány až logickým kompozičním systémem, v tomto případě právě technikou riffology.[79]

2.2.2 No Man's Sky (2016)

Případ hudby *No Man's Sky*[80] je v principu podobný tomu v *Ballblazer*. Zvukový tým nechal vytvořit a nahrát post-rockovou kapelu 65daysofstatic celé nové album, které sloužilo jako

[76] WOOLLER, Rene., BROWN, R. Andrew, et al. *A framework for comparison of processes in algorithmic music systems*. Generative Arts Practice, Sydney, Creativity and Cognition Studios Press, 2005. s. 109–124
[77] COLLINS, Karen. An Introduction to Procedural Music in Video Games. *Contemporary Music Review* [online]. 2009, **28**(1), 5-15 [cit. 2023-06-30]. ISSN 0749-4467. s. 8–9. Dostupné z: doi:10.1080/07494460802663983
[78] *Ballblazer* [videohra]. Tvůrci: David Levine – Peter S. Langston. Lucasfilm Ltd., 1984.
[79] BERNDT, Axel. Adaptive Game Scoring With Ambient Music. In: Monty Adkins – Simon Cummings (eds.). *Music Beyond Airports. Appraising Ambient Music.* Huddersfield: University of Huddersfield, 2019. s. 211. Dostupné z: https://www.researchgate.net/publication/335034399_Adaptive_Game_Scoring_with_Ambient_Music [cit. 30.12.2021]
[80] *No Man's Sky* [videohra]. 2016. Hello Games.

základ pro samotný soundtrack hry. Jednotlivé hudební elementy z alba pak vystřihali, a tak místo celých písniček měli k dispozici krátké zvukové stopy. Dále vytvořili svůj vlastní (veřejně nedostupný) hudební systém *Pulse* propojený s audio middlewarem Wwise. Tento hudební engine je v podstatě přehrávač náhodných souborů,[81] který ale reaguje na různé parametry ze hry – na to, v jaké oblasti se hráč nachází, ale třeba také na to, jak vzdálená je hráčova postava od určitého objektu ve hře. Jednotlivé hudební elementy jsou často tak krátké, že opět dává smysl mluvit spíše o generativní hudbě než o resekvencování.[82]

2.2.3 Spore (2008)

Spore[83] je jediný zde uvedený příklad, který zcela naplňuje definici generativní hudby ve hrách – jedná se autonomní systém a lze ho chápat jako generování nových dat (viz generativní algoritmus na začátku podkapitoly), nikoliv jako pouhou transformaci již existujících dat. Hudbu pro tuto hru složil Brian Eno a Peter Chilvers. Ta je generována na základě upravené verze vizuálního programovacího rozhraní Pure Data[84], která je propojená s herním enginem. Tento hudební engine vyvinul hudební programátor Aaron McLerran a Kent Jolly.

Hra hravým způsobem simuluje evoluční cestu živočišného druhu, který hráč pomáhá vytvořit během hraní. Na začátku hráč ovládá pouze mikroskopický organismus, který se během hry vyvíjí až do stavu, kdy cestuje mezi hvězdami napříč galaxií. Hra je rozdělena do různých ér, každá s odlišnou hudbou. Hudba však také reaguje na volby hráče. Například během hry si hráč vybírá, jak bude jeho druh vypadat a jaké budou jeho schopnosti přidáváním a upravováním jeho tělesných částí. To ovlivňuje i hudbu, např. podle toho, zda je druh považován za agresivnější nebo mírumilovnější v závislosti na jeho statistikách. Kvůli omezenému výkonu počítačů tehdejší doby používá *Spore*, podobně jako později *No Man's Sky*, již hotové zvuky. Ty jsou omezeny na jednotlivé tóny. Systém umožňuje práci i s dalšími DSP efekty.[85]

---

[81] MONGEAU, Anne-Sophie. Behind the Sound of No Man's Sky: A Q&A with Paul Weir on Procedural Audio. In: *A Sound Effect*. [online] 2017. [cit. 26-6-2023] Dostupné z: https://www.asoundeffect.com/no-mans-sky-soundprocedural-audio/

[82] WEIR, Paul. *The Sound of No Man's Sky*. Game Developers Conference, 2017. Dostupné z: https://www.gdcvault.com/play/1024067/The-Sound-of-No-Man [cit. 26-6-2023]

[83] *Spore* [videohra]. 2008. Maxis. Electronic Arts.

[84] *Pure Data* [software]. Open source. Originální distribuce: Miller S. Puckette. Dostupné z: https://puredata.info/ [cit. 26-6-2023]

[85] Kent Jolly – Aaron McLeran. *Procedural Music in SPORE*. Game Developers Conference, 2008. Dostupné z: https://www.gdcvault.com/play/323/Procedural-Music-in [cit. 23-1-2022]

3. Experiment: generování hudby na základě analýzy obrazu hry

3.1 Cíl experimentu

Jak již bylo zmíněno v úvodu, cílem experimentu bylo vytvořit jednoduchou počítačovou hru, či spíše její prototyp. Hudba této hry měla být generována v reálném čase na základě analýzy obrazu hry. To znamená, že hudba by se měla měnit v závislosti na pohybu a otáčení hráče ve hře, přičemž kamera by měla být vždy z první osoby.

Od počátku práce na experimentu bylo cílem, aby se hudba generovala na základě tří parametrů:

- Jaké objekty hráč vidí. Pojmem objekty jsou myšleny reálné předměty ve hře, např. hrnek nebo židle, ale třeba také stěna nebo strop.
- Jaká část konkrétního objektu je nazírána. Tedy aby např. podlaha v jedné místnosti mohla generovat na různých místech odlišnou hudbu.
- Jak daleko daný objekt je. Na základě tohoto parametru by se mohly např. filtrovat různé hudební vrstvy, podobně jako u vertikálního remixu.

Cílem experimentu bylo, aby byl zvuk syntetizovaný v reálném čase, nikoliv ovšem v „retro midi" kvalitě, ale v kvalitě připomínající klasicky připravený soundtrack. Přestože toto znamená značnou náročnost na výkon CPU, cílem bylo otestovat princip. Současně se počítalo s tím, že výkon procesorů stále roste, a tudíž by tento systém generování hudby mohl být uplatnitelný v budoucnu ve hře, která sama nebude na procesor příliš náročná. Hra, která by tímto způsobem generovala hudbu, by pravděpodobně od začátku byla vyvíjena s ohledem na náročnost. Tento systém by totiž pravděpodobně nebylo možné implementovat dodatečně – přístup k hudbě generované na základě analýzy obrazu by vyžadoval designování hry i konkrétních levelů již s ohledem na hudbu.

Krátce po započetí projektu (či spíše už při delší úvaze nad ním) začalo být zřejmé, že bude nutné vyřešit problém tzv. mickey-mousingu.[86] Dalším cílem experimentu tedy bylo najít takové technické řešení, které by skladateli umožnilo tvořit tak, aby hudební výsledek působil organicky, a aby měl možnost zamezit tomuto nežádoucímu efektu.

Posledním cílem bylo vytvořit tento systém generování hudby spolu s přehledným uživatelským rozhraním. Systém by tedy měl fungovat jako middleware zaměřený pouze na hudbu. V tomto middleware by mohl autor hudby snadno volně tvořit bez potřeby dalšího

[86] Mickey Mousing je filmová technika pocházející z raných filmů Walta Disneyho. Odkazuje na přesnou synchronizaci mezi hudbou a akcí na plátně, což může působit neúmyslně vtipně.

programování. Měl by být také automaticky propojený se hrou a umožnit tak skladateli zkoušet různá nastavení přímo za běhu hry, podobně jako umožňuje v této práci popsaný FMOD. Díky tomu by mělo být možné jednoduše v reálném čase upravova, jakým způsobem hudba na hru reaguje.

3.2 Technické řešení

Technické řešení experimentu lze rozdělit na dvě hlavní složky: analýzu obrazu a samotný hudební engine. Hra byla vytvářena v herním enginu Unity. V herním enginu také probíhala samotná analýza obrazu. Výsledky analýzy byly posílány přes komunikační protokol OSC[87] do hudebního enginu, který byl vytvářen v Max.[88]

3.2.1 Analýza obrazu

Jak bylo zmíněno, hlavním cílem bylo získat z analýzy obrazu tři informace: jaké objekty může hráč na obrazce vidět, jak daleko každý objekt je, a o jakou část objektu se jedná. Prvního cíle je dosaženo pomocí úpravy dodatečného barevného parametru materiálu daného meshe[89] přes skript v Unity. Všechny materiály využívají vlastní shader, který zahrnuje tato dodatečná data a také „průchody shaderem“[90] s vlastním pojmenováním (LightMode Tag). Při načítání scény ve hře je všem meshům přiřazena unikátní barva, která je pak identifikována při analýze. Tato vlastnost je v rámci experimentu nazývána "tagy". Všechny meshe také mají přiřazenou barvu na základě svých krajů modelu – červenou pro osu X, zelenou pro osu Y a modrou pro osu Z. Tato vlastnost je implementována v shaderu materiálu, a je označována jako "gradienty". Gradienty tímto způsobem umožňují zjistit, která část objektu je aktuálně zobrazována, resp. která část objektu je nejblíže. Vzdálenost objektu je také počítána v shaderu – k tomu je využita tzv. „mapa hloubky“ (depth map).

Všechny tyto parametry (tagy, gradienty a vzdálenost) jsou zobrazeny prostřednictvím tří různých kamer, které vykreslují do tří různých textur. Vykreslování probíhá každou půl vteřinu a textury mají čtvrtinu původního rozlišení, aby se snížila náročnost na výkon. Každá kamera má nastavený odlišný LightMode Tag, který určuje, jaký průchod shaderem (shader pass) se má použít, a tedy zda se zobrazují tagy, gradienty nebo mapa hloubky. Z tohoto důvodu je aktuální řešení analýzy závislé na použití Universal Render Pipeline (URP) – Standard Render Pipeline

[87] Open Sound Control (OSC) je protokol pro komunikaci mezi počítači, zvukovými syntezátory a dalšími multimediálními zařízeními, který je optimalizován pro moderní technologie sítí. Byl poprvé vyvinut na Center for New Music and Audio Technologies Adrianem Freedem, Mathewem Wrightem a dalšími.
[88] *Max* [software]. Cycling 74’. Dostupné z: https://cycling74.com/products/max
[89] V počítačové grafice je pojmem „mesh“ označován geometrický model objektu, složený z vrcholů, hran a trojúhelníků, které definují tvar a povrch objektu a umožňují jeho vizualizaci a interakci ve virtuálním prostoru.
[90] V angličtině „Shader pass“. Shader pass je jednou iterací nebo fází, kterou shader projde při vykreslování 3D objektů. Každý shader pass může obsahovat specifické instrukce, které ovlivňují, jak se objekt zobrazí, včetně nastavení osvětlení, textur, stínování a dalších efektů.

a High Definition Render Pipeline vlastnost LightMode nemají. Srovnání textur s tagy, gradientem, hloubkovou mapou a běžným renderem lze vidět na obrázku č. 5.

Samotná analýza je prováděna pomocí výpočetních shaderů (Compute Shader) – nejprve se zjistí všechny jedinečné barvy v textuře s tagy a poté se pro každou z těchto barev hledá nejsvětlejší bod (což znamená nejbližší bod) v mapě hloubky. V dalším kroku se na pozici nejsvětlejšího bodu hledá barva v textuře s gradienty. Nakonec se na základě barvy tagu získává informace o názvu příslušného meshe ze seznamu všech meshů v dané scéně a odesílá se do Max pomocí OSC v následujících formátech:

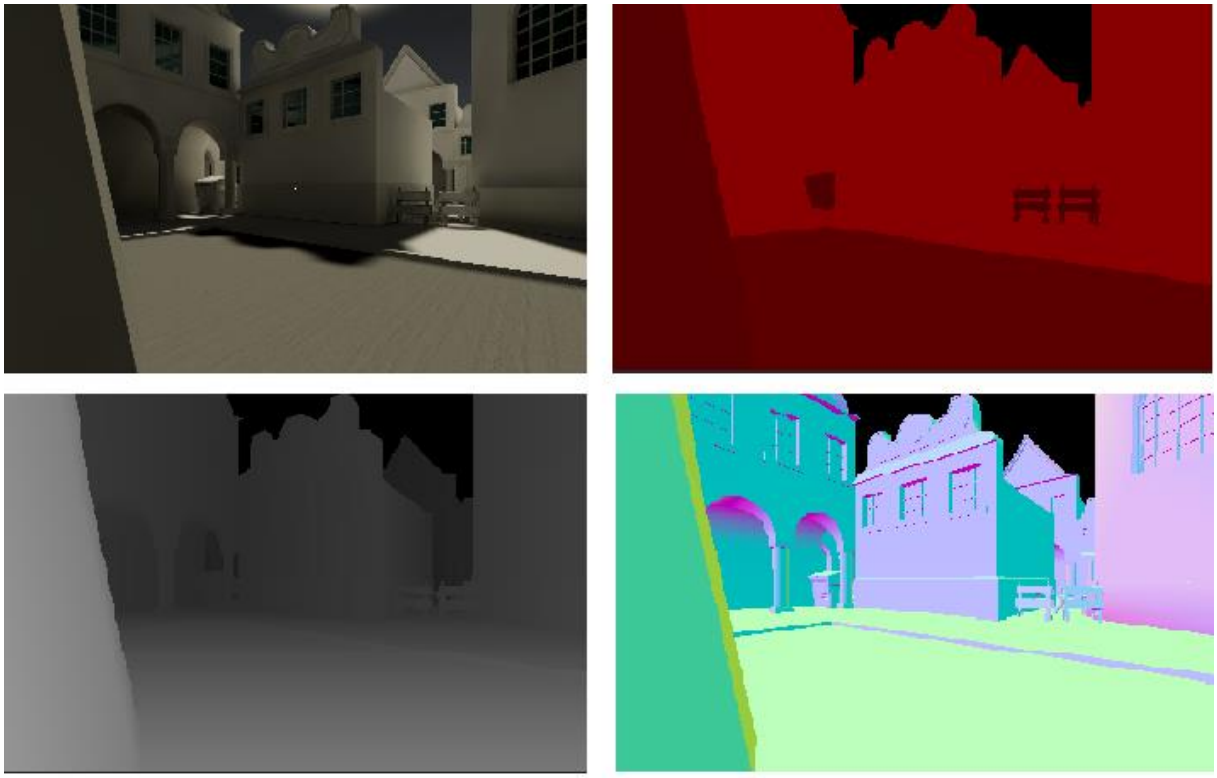
/nameOfMesh/dist: 0-255

/nameOfMesh/R: 0-255

/nameOfMesh/G: 0-255

/nameOfMesh/B: 0-255

Vzhledem k tomu, že většina kódu je psaná v běžně užívaném jazyce C#, ve kterém se v Unity pracuje, není tento kód k práci přiložen. V příloze 1 je ale možné si prohlédnout kód samotného výpočetního shaderu v jazyce HLSL, se kterým se lze setkat o poznání méně.



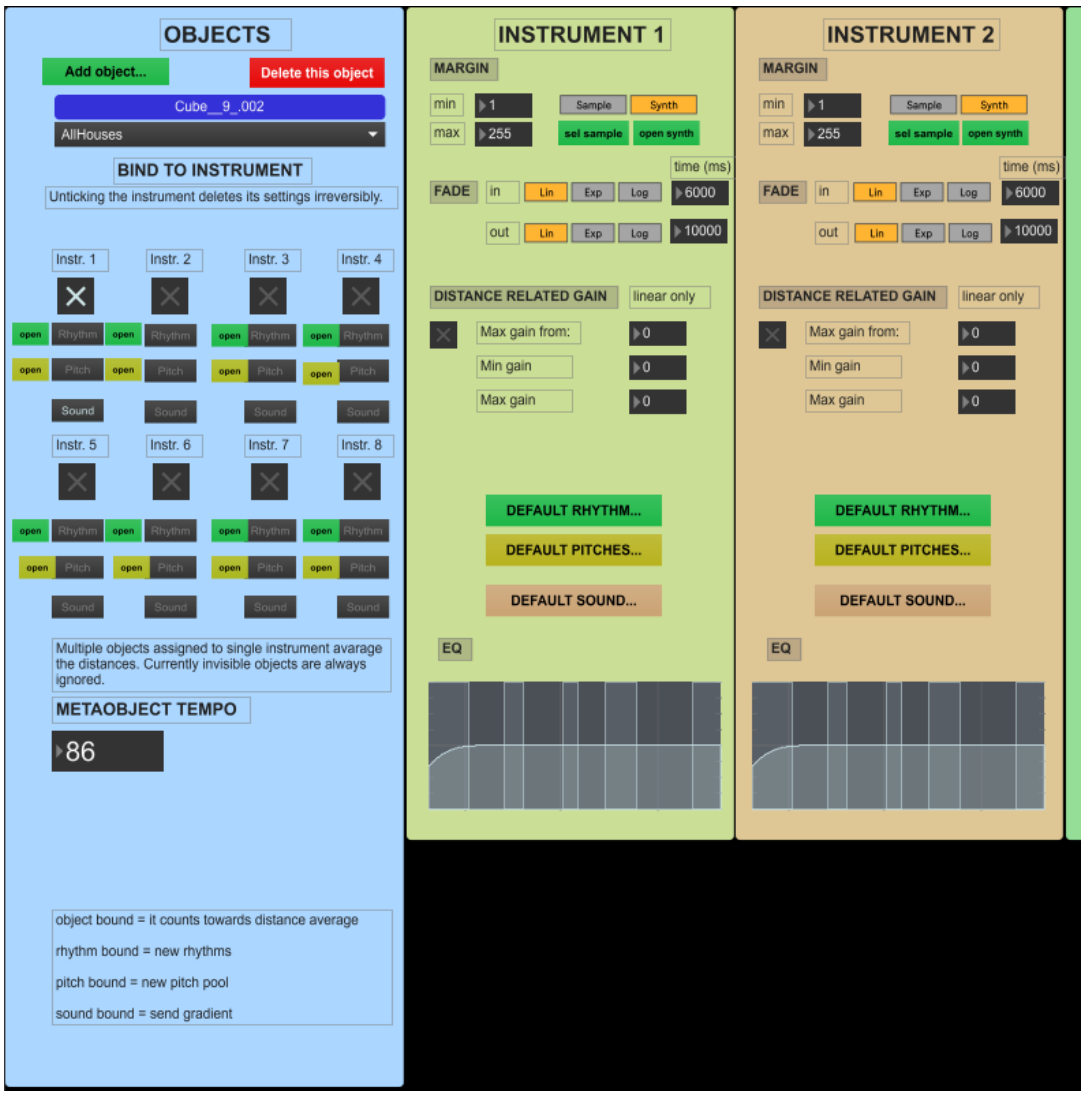
Obrázek č. 5

3.2.2 Hudební engine v Max

Vytvořené řešení nezahrnuje všechny funkce, které by měl dokončený hudební engine mít. Tyto nedostatky jsou popsány v následující kapitole 3.3. V téže kapitole je také probrána

nefunkčnost současného řešení. Některé příklady obsahují prozatímní grafiku a text, který na chybějící funkce odkazuje.

Jak již bylo zmíněno, jedním z cílů experimentu bylo najít řešení, jak se vyhnout efektu zvaný mickey-mousing. Z tohoto důvodu byl vymyšlen koncept „metaobjektů“, které jsou zásadní myšlenkou celého systému generování hudby. Každý metaobjekt (obrázek č. 6) se skládá z osmi nástrojů, přičemž každý nástroj může buď přehrávat zvolený sample, nebo syntetizovat zvuk v reálném čase. K jednomu metaobjektu je možné přiřadit až 32 různých objektů (tedy unikátně pojmenovaných meshů) ze hry. Pro každý objekt je dále možné nastavit, které z nástrojů má spouštět. Spuštění nástroje probíhá na základě vzdálenosti objektu. V případě, že je nástroji přiřazeno více objektů, počítá se vždy průměr vzdáleností všech aktuálně zobrazovaných objektů. Každý nástroj pak má nastavitelné rozpětí (min/max Margin), ve kterém má být spuštěn. Toto řešení umožňuje přidávat další vrstvy hudby na základě vzdálenosti objektu, a je obdobou techniky vertikálního remixu. Na základě vzdálenosti je také možné přidávat postupně hlasitost.



Obrázek č. 6

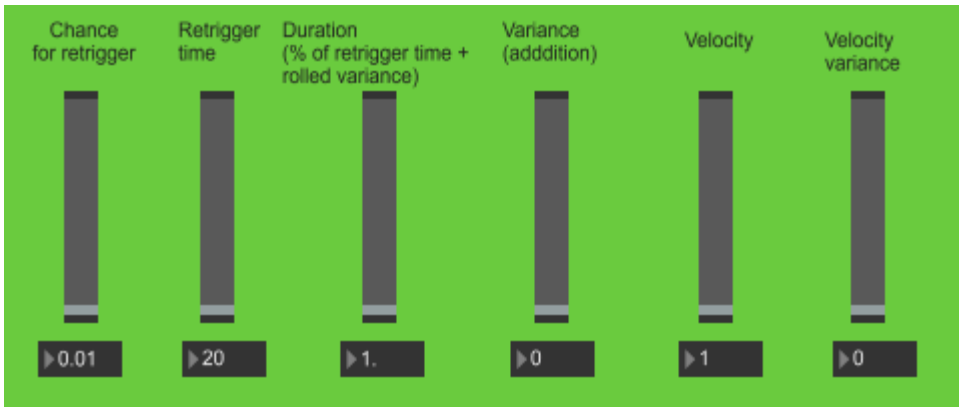
V nastavení nástroje lze najít také další možnost, jak omezit efekt mickey-mousingu – nastavení „Fade“. Skladatel může nastavit délku trvání fade in/out a jaký má mít tvar.[91] To slouží k tomu, aby si hráč přímo nespojoval konkrétní zvuk či hudební vrstvu s jedním objektem ve hře.

Jednotlivé nástroje mají vlastí nastavení defaultního rytmu, v případě syntezátoru také tónových výšek a zvuku. Jednotlivé objekty mohou tato nastavení také ovlivňovat, což bude vysvětleno podrobněji u každého z těchto nastavení.

3.2.2.1 Rytmus

Při nastavení defaultního rytmu nástroje si lze zvolit ze tří různých módů fungování: Continous, Random a Beat. Mód Continous pouze spouští či vypíná jeden tón, kterému se nemůže měnit tónová výška.[92] Mód Random má nastavení, které lze vidět na obrázku č. 7. Jednotlivé parametry mají tyto funkce:

- **Chance for retrigger** určuje šanci, že se spustí nota.
- **Retrigger time** udává čas v milisekundách, po kterém se má spustit další výpočet, který určí, zda se spustí nová nota.
- **Duration** je délka trvání noty. To je definované v procentech parametrů Retrigger time + aktuální náhodná hodnota Variance.
- **Variance** náhodně prodlužuje délku parametru Retrigger time. Nastavená hodnota znamená maximální možnou hodnotu, kterou může generátor náhodných čísel vygenerovat.
- **Velocity** udává hlasitost noty (v MIDI).
- **Velocity variance** náhodně zesiluje hlasitost noty (v MIDI). Stejně jako u Variance se vygeneruje náhodné číslo, které se pak sečte s hodnotou Velocity.



Obrázek č. 7

[91] Tlačítka „Exp“ a „Log“ ve skutečnosti neodkazují na přesně matematicky definované křivky, ale předem nastavené Bézierové křivky, které je tvarem trochu připomínají.

[92] Myšleno ve smyslu MIDI výšky, stále je možné zvýšit jeho frekvenci přes pitch shift, který ale spadá do nastavení zvuku.

Mód Beat (obrázek č. 8) pak umožňuje vytvářet rytmické paterny. K dispozici je nastavení až čtyř odlišných paternů pro každý nástroj, přičemž jejich pořadí může být postupné nebo náhodné. Každý patern je dělen na 16 a 24 (tedy jako trioly) impulsů. Patern si tedy lze představit jako jeden 4/4 takt, přičemž je možné individuálně nastavit šestnáctiny a šestnáctinové trioly. Každý impuls má nastavitelnou délku trvání noty, hlasitost a šanci, že se stane. Řešení Beat módu je z části inspirované aplikací *ReactionalMusic*, známé také jako *Gestrument*.[93]

Obrázek č. 8

Beat mód je první nastavení, které může být ovlivněné dalšími objekty v metaobjektu. V případě zapnutí tlačítka „Rhythm“ u daného nástroje (viz obrázek 6 – nastavení metaobjektu) je možné dále kliknout na tlačítko open. To otevře nastavení beat módu, které se podobá defaultním nastavení. Pro konkrétní objekt je tak možné mít vlastní nastavení. Podobně jako u spínání jednotlivých nástrojů, veškerá nastavení současně zobrazovaných objektů + defaultní nastavení se průměrují, a teprve z tohoto průměru se počítá, zda se jednotlivá událost stane. Kompozičně zajímavá je možnost nastavit u šance i záporné hodnoty. Díky tomu pak mohou jednotlivé objekty omezovat počet impulsů v paternu. Přibývající počet zobrazovaných objektů na obrazovce pak nemusí rytmus nutně pouze zahušťovat, ale může jeho části i zcela vypnout – je totiž možné nastavit tak vysoké záporné číslo, že po zprůměrování se šance rovná nule. I toto má sloužit jako nástroj omezující efekt mickey-mousing. Autorova hypotéza je, že při přidávání impulsů do paternu na základě jednoho objektu se tvoří výrazně silnější asociace mezi rytmem a objektem než při odebírání tohoto impulsu. Jinými slovy: s viditelným objektem je mnohem obtížnější asociovat absenci zvuku než skutečně znějící zvuk.

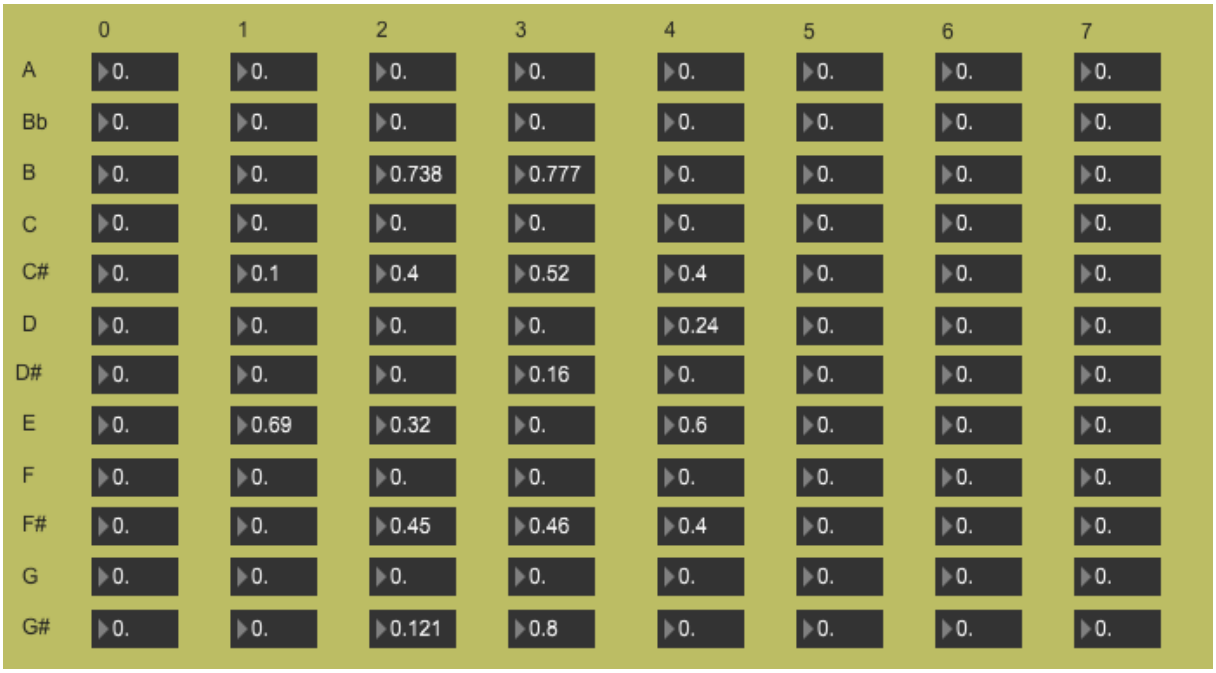
[93] *Reactional Music* [software]. Reactional Music. Dostupné z: https://reactionalmusic.com/

V rámci Beat módu je také potřeba zmínit nastavení tempa. Celý patch má jedno synchronizační (globální) tempo a každý metaobjekt má nezávisle na něm své vlastní tempo. Spuštění jednotlivých nástrojů je v tomto módu pozdrženo, dokud se obě tempa „nesejdou“ v jeden moment. Tímto řešením je tak možné dosáhnout toho, že nástroje uvnitř metaobjektu jsou vždy v synchronizaci nejen mezi sebou, ale také s dalšími metaobjekty, které mohou být v jiném tempu. Je ale třeba myslet na poměry jednotlivých temp, protože při komplikovanějších poměrech může spuštění trvat příliš dlouhou dobu.

3.2.2.2 Tónové výšky

Řešení tónových výšek (tedy melodie a harmonie) není žádným způsobem komplexní. Každý nástroj je omezený na jednohlas, resp. tóny jednoho nástroje přes sebe mohou přeznívat, ale nemůže v rámci jednoho nástroje zaznít více tónů současně. Celý vývoj směřoval spíše k ambientní dronové hudbě, a vícezvuky byly zamýšleny jako odlišné vrstvy (a tedy i nástroje). Plánovaná povaha hudby rovněž nevyžadovala komplexní řešení organizace tónových výšek. I přesto by bylo dobré, kdyby systém pokročilejší techniky pro generování tónových výšek byly k dispozici – to je krátce rozebráno v kapitole 3.3.2.

Tónová výška noty se volí náhodně podle pravděpodobností tabulky (viz obrázek č. 9). Podobně jako u rytmu v beat módu, vedle defaultního nastavení může mít každý objekt své vlastní pravděpodobnostní nastavení, které se průměruje na základě aktuálně zobrazovaných objektů. Ve výsledku jde tedy o tónový výběr, přičemž každá tónová výška v konkrétní transpozici má určitou šanci na to, že bude vybrána.

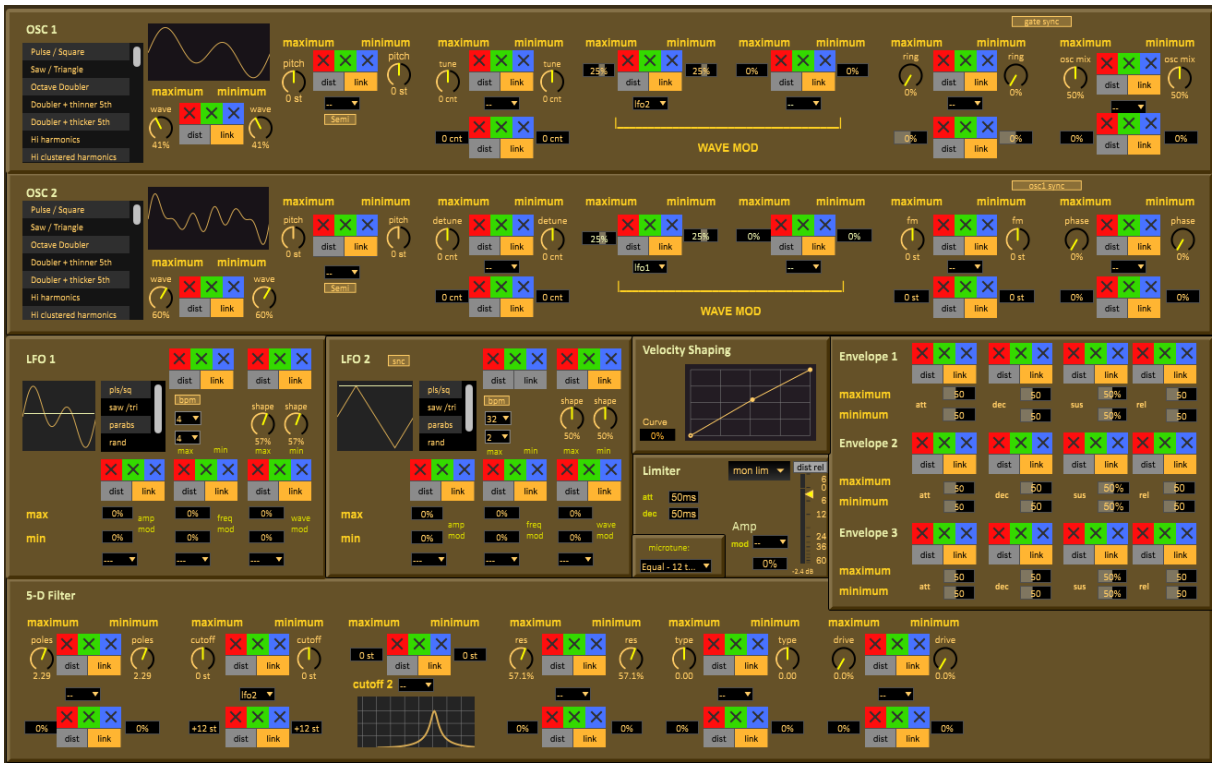


Obrázek č. 9

3.2.2.3 Barva tónu (nastavení syntezátoru)

Uživatelské rozhraní a možnosti syntezátoru vychází z již zmíněného syntezátoru *Husserl2*. Ten je ale upraven tak, aby jednotlivé parametry mohly reagovat na hodnoty přicházející ze hry, konkrétně na barvy gradientu a vzdálenost (obrázek č. 10). U barev gradientu je možné vybrat konkrétní barvu, v případě vícero barev se hodnota průměruje. Průměr zvolených barev gradientu anebo vzdálenost jsou pak kontrolní hodnotou pro daný parametr syntezátoru. U každého parametru je možné nastavit jeho minimální a maximální hodnotu. Výstupní hodnota je pak lineární interpolací kontrolní hodnoty (vždy v rozsahu 0–255) a minimální a maximální hodnoty daného parametru syntezátoru.

Tento systém má umožnit proměnu barvy na základě toho, na jakou část objektu hráč dívá (resp. která je nejblíže), popř. na základě toho, jak je objekt daleko. Pro každý objekt lze individuálně nastavit, zda má hodnotami barev gradientu a vzdáleností přispívat k barvě vybraného nástroje či nikoliv, podobně jako u rytmu a tónových výšek. Kromě nastavení syntezátoru má pak každý nástroj k dispozici svůj vlastní ekvalizér, který je ale statický.



Obrázek č. 10

3.2.2.4 Mix

Této složce nebyla příliš věnována pozornost. Mix je v rámci hudebního enginu vyřešen tak, že se signály pouze sčítají a dělí počtem zdrojů. To je řešeno za pomocí Maxovských objektů „mc“, které umožňují jednoduchou správu multikanálového audia. Kromě toho má každý nástroj má dispozici vlastní nastavení hlasitosti, resp. v každém nastavení syntezátoru lze nastavit také gain.

3.3 Reflexe zvolených řešení

3.3.1 Analýza obrazu

Zvolené řešení analýzy obrazu funguje po technické stránce velmi dobře. Díky využití shaderů probíhá výpočetně náročná část na grafické kartě, která je na takovéto výpočty speciálně stavěná, a tak je dopad na výkon v podstatě zanedbatelný. Technické nedostatky současného řešení lze shrnout v těchto bodech:

- Bylo by možné přidat další funkci, která by počítala, kolik konkrétní objekt zabírá obrazovky. Tato informace by mohla být dále zpracována v hudebním enginu. Lze si představit, že v určitých kontextech hry by dávalo smysl hudebně reflektovat např. to, že když objekt–hrnek zabírá 60% obrazovky, jedná se o důležitější „událost", než když 60% obrazovky zabírá objekt–stěna.
- Současné řešení nenabízí možnost, jak zjistit, kde na obrazovce se konkrétně bod nachází. Vysvětlit přesně důvod proč by vyžadovalo podrobně popsat celý postup analýzy, což by zabralo neúměrně mnoho textu, který by se už příliš odkláněl od tématu práce. Krátce to lze shrnout tak, že důvodem je paralelnost výpočtu v mnoha vláknech, které plyne z fungování grafické karty a výpočetních shaderů.
- Řešení v této podobě nedokáže analyzovat terén a částicové efekty v Unity, což by mohlo být případně potřeba.
- Pro gradienty jsou využity tzv. vertexové barvy[94], které netvoří zcela plynulý gradient. Tento problém je zanedbatelný v případě, že je plocha členitější meshe (obsahuje více trojuhelníků). V některých případech však výsledek není zcela uspokojivý. Závisí tedy i na tom, jak je mesh vymodelován. Současně s tím autora nenapadá lepší řešení, které by bylo v herním enginu automatizované a současně nebylo výpočetně náročné. Přesvědčivější gradienty je teoreticky možné vytvořit jako alternativní textury přímo v modelovacím programu, který umožňuje programovatelné generování textur, jako např. *Blender*.[95] To však klade další nároky na tvůrce meshů (resp. již otexturovaných objektů), a současně je i znatelně náročnější na výkon.

Kromě těchto technických nedostatků se při práci na experimentu také ukázalo, že může být problematické komunikovat požadavky při přípravě meshů a scén v herním enginu. Je totiž nutné zvolit jeden konzistentní postup práce. Při experimentu spolupracoval autor analýzy obrazu a hudebního enginu se dvěma herními designéry, kteří vytvářeli testovací herní levely. Jeden z designérů však vytvořil v Unity scény, které nebyly příliš kompatibilní

[94] To znamená, že každý bod v geometrii meshe má svou barvu.
[95] Blender [software]. Open source. Dostupné z: https://www.blender.org/

s naprogramovanou distribucí tagů. V jednom případě byla velká část „objektů“ součástí jednoho velkého herního objektu, podstatná část scény tedy byla vytvořena v modelovacím programu Blender, a nikoliv v Unity samém. Při vývoji běžné hry by toto pravděpodobně nebyl problém, ovšem systém distribuce tagů nedokázal pracovat s jedním takto hierarchizovaným objektem. Skript distribuce tagů byl přitom takto vytvořen záměrně: u herních objektů je běžné, že např. dveře jsou sice odlišný herní objekt než dům, hierarchicky ale spadají pod dům.[96] Autor si však přál, aby byl herní objekt naopak vždy vnímán jako jeden celek. V případě práce na podobném projektu ve více lidech by tedy bylo nutné předem přesně stanovit, jakým postupem má probíhat příprava scén a meshů, a případně změnit skript distribuující barevné tagy.

3.3.2 Hudební engine

3.3.2.1 Max jako hudební engine

Nápad vytvořit hudební engine za pomocí Max se ukázal jako špatný z několika důvodů, které nakonec vedly k nezprovoznění celého hudebního enginu, a tak ani k jeho řádnému otestování. Max byl k realizaci experimentu původně zvolen ze tří důvodů:

1. Jedná se komplexní program, který nabízí spoustu možností, jak pracovat se zvukem.
2. Umožňuje snadno a rychle vytvářet uživatelské prostředí.
3. Max jako program byl autorovi známý a měl s ním již poměrně dost zkušeností. Současně měl autor v době započetí projektu pouze minimální znalosti programování v kódu.

Některá rozhodnutí v designu se ovšem ukázala jako vysoce problematická a zůstává i otázkou, zda je řešení přes Max vůbec vhodné (autor se přiklání k tomu, že nikoliv). Zvolené řešení vedlo k velmi komplexnímu patchi, a reakce programu se postupně stále zpomalovala. Některé změny již trvaly desítky vteřin, což dále zpomalovalo samotný vývoj a opravování chyb. Problém ovšem nebyl pouze v tom: zpomalená reakce se přímo dotýkala i uživatelského rozhraní. Výsledek se tak začal míjet s jedním z cílů experimentu – vytvořit snadno ovladatelné rozhraní, ve kterém lze jednoduše měnit parametry generování hudby za běhu samotné hry. Pouhé přidání a otevření nastavení jednoho objektu trvalo několik minut, pokud tedy celý Max rovnou nespadl.

Důvody této nestability nejsou autorovi zcela zřejmé[97], nicméně na základě různých fór a diskusí dospěl k přesvědčení, že hlavní problém současného řešení spočívá v nadměrném

[96] Tedy dům je tzv. „parent“ a dveře „child“.

[97] I při vypnutém DSP (kterého by se to ani dotýkat nemělo) aplikace zkrátka na delší dobu zamrzávala, přitom si brala pouze malou část výkonu procesoru. Podle Windows programu Sledování prostředků nebylo přitom nijak výrazně vytížené pouze jedno vlákno procesoru, problém tedy nebyl v tom, že by Max neuměl využívat všechna vlákna procesoru.

užívání abstrakcí[98] (patch je poměrně jasně hierarchizován) a generováním nových Maxovských objektů.[99] Přidávání a nastavování objektů v metaobjektu je řešeno za pomocí generování nových Maxovských objektů, často také i mnoha různých abstrakcí. Další významné zpomalení by mohla způsobovat databáze, která je rovněž řešena čistě v Max za pomocí Maxovského objektu „pattrstorage". S každým novým objektem v metaobjektu se generuje mnoho nových Maxovských objektů „pattr", kterými se do databáze přidávají nové položky.

Odlišným řešením databáze v Max by mohlo užití *SQLite*[100] databáze. Autor se pokusil použít objekty z IRCAMovské knihovny *FTM*[101], která je zaměřená na datové struktury právě s využitím SQLite. Setkal se ovšem s problémem, že zatímco na jednom ze dvou odlišných zařízeních[102] se stejným operačním systémem všechny objekty z knihovny fungovaly, na druhém už některé ne – a to včetně zásadního objektu, který zprostředkovával komunikaci s databází. Toto řešení se tedy ukázalo jako nepoužitelné. Bylo by tedy nutné vytvořit vlastní správu této databáze a veškeré požadované funkce buď jako Maxovské objekty v C, nebo vyřešit komunikaci s databází přes JavaScript. Kromě toho by ale bylo zapotřebí vymyslet jinou správu dat, která je na generování Maxovských objektů také postavena.

Samotný vývoj v Max se nakonec ukázal jako mimořádně neefektivní – v některých případech je bylo nutné přidávat obrovské množství Maxovských objektů pro funkci, kterou by šlo snadno vyřešit poměrně jednoduchým kódem. V některých případech si autor takto i pomohl vlastními skripty v JavaScript, ne vždy to ale bylo možné. JavaScript totiž nativně nepodporuje tvorbu matic (vícedimenzionální „arrays"), se kterými by bylo potřeba provádět různé matematické operace. Import matematických knihoven[103] pro JavaScript v Max je opět problematický, a vyžadoval by najít nějaké řešení přes *Node.js*, přičemž si autor není jist, zda by to skutečně bylo možné.

Z výčtu těchto problémů by mělo být zřejmé, z jakého důvodu se autor přiklání k tomu, že Max je nevhodné technologické řešení tohoto experimentu. Po setkání se s těmito problémy si autor udělal další průzkum, aby navrhl odlišná řešení. Všechna tato řešení vždy v nějaké míře obsahují programování v kódu. Jak se však ukazuje, to samé by bylo nutné i v případě nového řešení za pomocí Max. Optimalizace v Max je rovněž problematická. Prostředky procesoru lze sice v Max optimalizovat za pomocí objektu „poly~", autor už si ale není vědom možnosti

[98] Abstrakce jsou Maxovské patche, které jsou uložené na disku a lze je používat jako Maxovské objekty. Zároveň umožní snadnější práci s argumenty.
[99] Pro přehlednost: objekt v metaobjektu odkazuje na objekt ve hře, resp. na jeho mesh; Maxovský objekt je součást fungování programu Max: každý objekt reprezentuje nějakou funkci.
[100] *SQLite* [software]. D. Richard Hipp. Dostupné z: https://www.sqlite.org/index.html
[101] *FTM* [software]. IRCAM. Dostupné z: https://forum.ircam.fr/projects/detail/ftm/
[102] Konkrétně šlo o notebook a stolní počítač s tehdy aktuální verzí Windows 10.
[103] Např. tato knihovna by byla řešením:
*Math.*js [software]. Jos de Jong. Dostupné z: https://mathjs.org/

optimalizace, co se týče operační paměti. Té přitom současné řešení v Max spotřebovává nepřijatelně mnoho (více než 1 GB), a to je bez nahraných samplů.

3.3.2.2 Koncept

Ačkoliv nebylo možné celý experiment otestovat řádně, některé složky šlo v průběhu vývoje vyzkoušet zvlášť. Autor zjistil, že účinným opatřením proti efektu "mickey-mousingu" je práce s délkou trvání fade in/out a tvarem křivky. Například deseti vteřinový fade in s exponenciálním tvarem v praxi fungoval jako jakési zpoždění, a asociace zvuku se zobrazovaným objektem se (z autorova pohledu) utvářela mnohem méně.

Dále bylo možné v rané fázi vývoje vyzkoušet proměnu barvy jednoho tónu. Ten měl v podstatě roli dronu. Barva tónu se proměňovala na základě barev gradientu jednoho objektu, konkrétně se jednalo o podlahu místnosti. Ačkoliv je toto těžké hodnotit, autorův dojem z výsledku byl vcelku pozitivní. Zdálo se, že se jedná o prvek s velkým potenciálem, který může generativní kompozici pro virtuální prostor značně oživit. Užití tohoto prvku by mohlo být zajímavé také v kombinaci s level designem hry – např., pokud by se měl hráč z herních důvodů přesouvat primárně z jedné strany velké místnosti či oblasti na druhou, bylo by možné tento pohyb jasně (a přitom plynule) reflektovat právě proměnou barvy tónu (dronu). Například zvyšující se zkreslení (distortion) takového dronu by mohlo postupně podkreslovat zvyšující dramatičnost ve hře, vycházející právě z level designu.

K poskytnutí dalších postřehů by bohužel bylo nutné otestovat celý, již funkční hudební engine. I když bylo možné odděleně testovat proměnlivost tónových výšek na základě zobrazovaných objektů (s různými šancemi pro každý objekt) nebo proměnlivost rytmu v beat módu (včetně negativních hodnot pro šanci), není možné dospět k závěru o jejich konceptuální nosnosti, pokud nejsou začleněny do komplexnější hudební struktury.

3.3.2.3 Koncepční nedostatky

Koncepčními nedostatky jsou míněny v současnosti nevhodné nebo nedořešené koncepční řešení. Nejde o tedy konkrétní technické zpracování, ale o to, jaké možnosti by měl hudební engine obsahovat. To vychází z autorova pohledu jakožto skladatele, přičemž je brán ohled na to, aby realizace těchto možností byla realistická. Největší nedostatky současného řešení hudebního enginu lze shrnout v těchto bodech:

- **Práce se samply**: raná verze obsahovala jednoduchou možnost přehrávání samplů, ale kvůli pozdější restrukturalizaci celého hudebního enginu tato funkce přestala fungovat. Jednoduchý systém přehrávání samplů by ale pravděpodobně nebyl dostačující. Jako ideální řešení se zdá vytvoření systému, který by se podobal multinástrojům ve FMOD (viz kapitola 1.4.2.2 Vertikální remix). V takovém systému by bylo možné přidat jednomu nástroji v metaobjektu více různých samplů, které by se mohly přehrávat buď

náhodně, anebo v pořadí. V případě náhodného přihrávání by pak mohly být šance přepočítávány stejně jako je to již u tónových výšek.

- **Dynamické nahrávání/odebírání z paměti:** Hudební engine byl již od počátku designován s ohledem na to, aby bylo možné dynamicky přidávat a odebírat z paměti potřebné samply. Stejně tak byl design navržen tak, aby bylo možné zapínat a vypínat DSP jednotlivých metaobjektů podle potřeby.[104] Reálná implementace ale chybí.
- **Rytmus nástrojů – beat mód:** Současná verze nabízí poměrně dost možností, které je teoreticky možné dále škálovat (delší takty, více paternů k nastavení). Má však dvě velká omezení: limitaci na 4/4 takt a možnost volit si pouze mezi náhodným anebo cyklickým přehráváním paternů. Nepochybně by bylo užitečné, kdyby mohl skladatel svobodně navolit různá opakování patermů. Tedy například, že chce, aby se třikrát přehrál patern 1, následovně jednou patern 2, poté třikrát patern 3, a na závěr patern 4 (3x1, 1x2, 3x3, 1x4), přičemž tento vzorec se má opakovat. To by šlo dále rozšířit o to, že by se dalo vypsat těchto vzorců více, a ty by se pak mohly buď cyklit, či být vybírány náhodně. Náhodné přehrávání by přitom mohla ovlivňovat i data z analýzy obrazu. Dalším nedostatkem je určité omezení v trvání jednotlivých not, které je vždy stejné. To sice může stačit při užití beat módu jako „drum machine“, při tvorbě melodie už ale může být současné řešení omezující.
- **Tónové výšky:** Tomuto aspektu nebyla od začátku příliš věnována pozornost – mimo jiné i proto, že se jedná o poměrně komplexní problém, především má-li vytvořený systém nabízet dostatečné množství možností. Vzhledem k tomu, jakým způsobem se pracuje v jiných systémem s pravděpodobností, tak by se autor pravděpodobně přikláněl k nějakému řešení za pomocí Markova modelu, ve kterém by byly váhy navíc ovlivňovány aktuálně zobrazovanými objekty ve hře. Vzhledem k možnosti strukturovaného rytmu (beat mód) by přicházelo v úvahu zkombinovat Markovův model s řešením inspirovaným modelem generativní gramatiky.[105]
- **Zvuk:** Bylo velmi nápomocné mít k dispozici již hotový syntezátor v Max, který se pouze upravil pro účely experimentu. Syntezátor ale neobsahoval všechny možnosti, které by autor potřeboval, přičemž asi nejvýraznější omezení byla s umístěním zvuku v prostoru. Syntezátor neobsahoval napojení panoramy na nízkofrekvenční oscilátor, který přitom

[104] Tato je prakticky dána aktuálně načtenou scénou. V Unity je hra členěná na jednu či více různých scén, přičemž pouze aktuálně načtené scény zabírají systémové prostředky. Jednotlivé scény lze asynchronně nahrát a odehrávat z paměti. Stejným způsobem byl zamýšlen i zvukový engine – při změně scény by se z paměti mělo odehrát vše, co není potřeba, a ušetřit tak systémové prostředky.

[105] Více informací o těchto, a i dalších modelech lze najít v:
LOPEZ DUARTE, Alvaro E. Algorithmic interactive music generation in videogames. *SoundEffects - An Interdisciplinary Journal of Sound and Sound Experience* [online]. 2020, **9**(1), 38-59 [cit. 2023-06-30]. ISSN 1904-500X. Dostupné z: doi:10.7146/se.v9i1.118245

ovládá i jiné parametry syntezátoru. Dokončený hudební engine byl měl mít takovou funkci k dispozici. Rovněž by bylo dobré najít uživatelsky přístupné řešení pro nastavení ekvalizace reagující na data z analýzy obrazu.[106]

- **Mix:** Toto je další nedořešená část současného řešení. Koncept nepočítá s kombinací hudby a zvukových efektů ze samotné hry. Možným řešením by bylo posílat do hudebního enginu informaci o tom, jak hlasitá má hudba být, což by v praxi byl multiplikátor síly signálu již zmixované hudby. To by ale bylo možné nejspíše pouze s jednoduchým řešení zvukových efektů přímo v herním enginu, a ne při za použití dalšího middleware jako FMOD nebo Wwise. Dále není dostačující ani současná podoba mixu v hudebním enginu. Ta pouze sčítá signály a dělí je jejich počtem. Bylo by dobré mít k dispozici možnost, jak přidělit celým metaobjektům odlišnou hlasitost. Přístup k jednotlivým hlasitostem nástrojů uvnitř metaobjektů je rovněž komplikovaný. Za ideální řešení bychom nejspíše mohli považovat zvláštní uživatelské rozhraní, ze kterého je možné všechny parametry hlasitostí snadno měnit. To by mělo také urychlit samotnou tvorbu, kdy je potřeba hudební mix testovat za běhu hry.
- **Propojení se hrou:** Současně navržené řešení, ve kterém je hudební engine samostatnou aplikací spuštěnou na pozadí, je sice teoreticky možné, přináší však vlastní problémy. Nepochybně by bylo lepší, kdyby byl hudební systém integrován do herního enginu. Aplikace postavená v Max totiž nikdy ve skutečnosti neběží jako proces na pozadí – je jí možné pouze částečně skrýt. Hráč hry by tedy mohl omylem hudební engine nezávisle na hře vypnout. Hudební engine by musel také obsahovat vypínací mechanismus, který by zaručil jeho vypnutí při případném pádu hry. Při dalším vývoji by pravděpodobně vyvstaly i další problémy.

3.4 Návrh jiných řešení

Po analýze toho, proč je realizace za pomocí Max problematická, se autor pokusil prozkoumat případná další řešení. Ta nejsou probádána detailně – jedná se pouze o některé směry, kterými by bylo možné se vydat. I tak lze ale odhadnout některé jejich výhody a nevýhody.

Obecně platí, že u všech těchto variant je možnost buď vytvořit zcela nový vlastní syntezátor, anebo použít open source syntezátor, který by se podle potřeb projektu upravil.[107] Úprava takového syntezátoru ale může být sama o sobě natolik časově náročná[108], že příprava nového

[106] Raná verze hudebního enginu tuto možnost umožňovala, nastavení ale bylo poměrně komplikované a zdlouhavé, proto ho autor v pozdějších verzích odstranil.

[107] Autor uvažoval konkrétně o syntezátoru Surge XT.
Surge XT [software]. Open source. Dostupné z: https://surge-synthesizer.github.io/

[108] Vyžaduje nejprve porozumět samotné architektuře softwaru, a dále pak vymazat nepotřebné funkce, což samo může vyžadovat další úpravy ostatních složek softwaru. A poté teprve přidat funkcionalitu potřebnou pro projekt.

syntezátoru by mohla být ve výsledku výhodnější. Navíc by byla od začátku vytvářena s ohledem na potřeby projektu, výsledek by tedy pravděpodobně byl přehlednější, a bylo by snazší do něj přidávat další funkce.

Je nutné zmínit, že autor nemá tak velkou znalost softwarového inženýrství, aby dokázal zcela dohlédnout důsledky navržených řešení. Bylo by nutné si prostudovat navržená řešení detailněji a dovzdělat se v tom, jak konkrétně by měla být realizována. Obecně si autor uvědomuje především mezery ve znalosti integrace hudebních (či obecně zvukových) enginů do herního enginu.

3.4.1 Fabric

Fabric je méně známý audio middleware, který se přímo integruje do prostředí herního enginu Unity. Jak autor zjistil (bohužel až po neúspěšném řešení v Max), Fabric má k dispozici rozšíření, které přidává možnost vytvářet vlastní syntezátor. Jeho tvorba by přitom mohla být ze všech navržených variant nejjednodušší – probíhá za pomocí propojování „nodů“, podobá se tak softwarům jako Max a PureData (obrázek č. 11).

Správa dat[109] by mohla být řešena skripty v samotném herním enginu. Pro jednodušší tvorbu by bylo potřeba vytvořit rozšíření samotného herního enginu, které by obsahovalo uživatelské rozhraní. Toto řešení by také vyžadovalo, aby byl skladatel schopný se orientovat v herním enginu.

Podstatné by bylo zjistit, jak moc je Fabric optimalizovaný. To může být obzvlášť důležité v případě zmíněných syntezátorů. Není jisté, zda by umožňoval distribuovat výpočty do různých vláken procesoru, především tedy mimo hlavní vlákno, na kterém probíhá mnoho výpočtů samotné hry.

Obrázek č. 11

[109] Tím je myšleno vybudování databáze a zpracování informací analýzy obrazu tak, že by se syntezátoru pouze „dodávaly informace“ o tom, jaký parametr má mít jakou hodnotu, jaký hlas má hrát, atd.

3.4.2 Integrovaný systém

Tato varianta se podobá řešení za pomocí Fabric. Místo Fabric by se ale vytvořil zcela nový systém přímo v Unity. DSP by pak mohlo probíhat za pomocí skriptů v C# přímo v Unity, anebo za pomocí naimportovaných DLL knihoven, které by byly napsané v efektivnějším (ale komplikovanějším) C či C++. Správa dat a uživatelské rozhraní by byly řešené stejně jako v předchozím návrhu využívajícím Fabric.

V případě DSP v samotném Unity je toto řešení potenciálně problematické, co se týče optimalizace. V Unity není zcela snadné rozvrhnout výpočty do vícero vláken – za tímto účelem obsahuje vlastní API, jejíž využití je ale zamýšleno pro trochu jiné účely. Je možné, že by implementace DSP s využitím více vláken byla bezproblémová, autor ovšem vnímá, že by při vývoji mohly vyvstat nové potíže.

V případě naimportovaných knihoven by bylo nejspíše nutné přesunout do nich i nějakou část správy dat, jako je třeba správa hlasů. Jinak hrozí podobné riziko jako v případě DSP v samotném Unity – v rozvržení výpočtů do vícero vláken procesoru by mohlo být problematické. Toto řešení se také začíná podobat poslednímu navrženému řešení.

3.4.3 Zprostředkovaný systém

Toto řešení se nejvíce podobá tomu, které bylo vytvořeno v rámci experimentu. Hudební systém by byl v tomto případě samostatným softwarem, který by byl propojený s herním enginem (pro komunikaci dat) a audio middlewarem[110], popř. by streamoval audio data zpět do herního enginu, pokud by hra nevyužívala audio middleware pro dynamický mix. Takové řešení je potenciálně jednodušší optimalizovat. Mohlo by navíc fungovat mimo samotný herní engine, a tak by nevyžadovalo po skladateli schopnost orientovat se v herním enginu.[111] Je otevřenou otázkou, nakolik by mělo být toto řešení integrované do herního enginu – tedy zda by mělo fungovat podobně jako FMOD, který je do Unity integrován a tvorba probíhá v samostatné aplikace FMOD Studio; anebo zda by mělo jít o paralelně běžící software (což je pravděpodobně snazší vytvořit), kontrolovaný hrou.

I přes potenciálně jednodušší optimalizaci se toto řešení jeví jako časově nejnáročnější, ale také jako nejspolehlivější.

[110] To je v podstatě identické řešení jako v případě *No Man's Sky*.

[111] To by ale stejně bylo vhodné, už jen proto, aby skladatel věděl, jaké jsou v dané scéně herní objekty. Šlo by ale o něco přístupnější řešení – požadavky na znalost herního enginu by byly nižší.

Závěr

V rámci této práce byla provedena analýza soundtracku pro videohry, s důrazem na generativní hudbu. Zvláštní důraz byl kladen na experimentální přístup generování hudby na základě analýzy obrazu hry. V rámci práce byl vytvořen experiment uplatňující tento přístup. Z důvodu technických obtíží však byla jen velmi omezená možnost tento přístup v rámci experimentu otestovat. I přesto ale bylo možné na celém procesu představit specifické problémy, které mohou nastat při generování hudby pomocí tohoto přístupu. Tyto problémy vyžadují další výzkum a vývoj, zejména v kontextu řešení problému mickey-mousingu a konkrétní technické realizace.

Tato práce rovněž otevírá dveře k dalšímu výzkumu v oblasti generování hudby na základě analýzy obrazu. Tento koncept se přitom nemusí omezovat pouze na videohry, ale může se stát základem pro řadu dalších aplikací, jako jsou umělecké VR projekty nebo technologie rozšířené reality. Představme si, že umělá inteligence analyzuje obraz reálného světa prostřednictvím kamer v brýlích rozšířené reality a generuje hudbu na základě těchto dat. Tento přístup může otevřít cestu ke zcela novým způsobům interakce s hudbou a zvukem.

Vývoj experimentu mě obohatil o cenné zkušenosti nejen v oblasti hudební kompozice, ale také v oblasti herního vývoje a softwarového inženýrství. Šlo o velmi naučnou cestu, která mě dovedla od minimálních dovedností v programování v kódu až po tvorbu vlastních shaderů v HLSL a práci v Unity v jazyce C#.

Navzdory výzvám, které generování hudby na základě analýzy obrazu přináší, jsem přesvědčen, že tento přístup má potenciál obohatit oblasti hudby, návrhu her a možná i dalších forem umění. Doufám, že tato práce pomůže dalšímu výzkumu v nepříliš prozkoumané oblasti generativní hudby ve videohrách.

Seznam použitých zdrojů

Použitá literatura a internetové zdroje

BERNDT, Axel. Musical Nonlinearity in Interactive Narrative Environments. In: *Proceedings of the International Computer Music Conference* [online]. August 16–29, 2009, Montreal, Kanada. Dostupné z: https://www.researchgate.net/publication/261803850_Musical_Nonlinearity_in_Interactive_NarrativN_Environments/references [cit. 20-6-2023]

BERNDT, Axel. Adaptive Game Scoring With Ambient Music. In: Monty Adkins – Simon Cummings (eds.). *Music Beyond Airports. Appraising Ambient Music.* Huddersfield: University of Huddersfield, 2019. s. 197–226. Dostupné z: https://www.researchgate.net/publication/335034399_Adaptive_Game_Scoring_with_Ambient_Music [cit. 30-12-2021]

BERNDT, Axel. HARTMANN, Knut. The Functions of Music in Interactive Media. In: Spierling, U., Szilas, N. (eds) *Interactive Storytelling.* ICIDS 2008. Lecture Notes in Computer Science, vol 5334. Springer, Berlin, Heidelberg. 978-3-540-89454-4. s. 4. Dostupné z: https://doi.org/10.1007/978-3-540-89454-4_19 [cit. 20-6-2023]

BROWN, Emily a Paul CAIRNS. A grounded investigation of game immersion. In: *CHI '04 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2004, 2004-04-24, s. 1297-1300. ISBN 1581137036. Dostupné z: doi:10.1145/985921.986048 [cit. 27-12-2021]

COLLINS, Karen. An Introduction to Procedural Music in Video Games. *Contemporary Music Review* [online]. 2009, **28**(1), 5-15 [cit. 2023-06-30]. ISSN 0749-4467. Dostupné z: doi:10.1080/07494460802663983

COLLINS, Karen. Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design. Cambridge, Mass: MIT Press, 2008. ISBN 978-0-262-03378-7

COLLINS, Nick. The Analysis of Generative Music Programs. In: *Organised Sound* [online]. 2008, **13**(3), 237-248 [cit. 2023-06-30]. ISSN 1355-7718. Dostupné z: doi:10.1017/S1355771808000332

DORIN, Alan. Generative processes and the electronic arts. In: *Organised Sound* [online]. 2001, s. 47-53 [cit. 2023-06-30]. ISSN 1355-7718. Dostupné z: doi:10.1017/S1355771801001078

ENO, Brian. London, UK: Faber, 1996. ISBN 0-571-17995-9.

HERBER, Norbert. *Amergent Music: behavior and becoming in technoetic & media arts*. Disertační práce. Plymouth: University of Plymouth, 2010. Dostupné z: https://pearl.plymouth.ac.uk/handle/10026.1/307 [cit. 27-12-2021]

JOLLY, Kent – MCLERAN, Aaron. *Procedural Music in SPORE*. Game Developers Conference, 2008. Dostupné z: https://www.gdcvault.com/play/323/Procedural-Music-in [cit. 23-1-2022]

LOPEZ DUARTE, Alvaro E. Algorithmic interactive music generation in videogames. *SoundEffects - An Interdisciplinary Journal of Sound and Sound Experience* [online]. 2020, **9**(1), 38-59 [cit. 2023-06-30]. ISSN 1904-500X. Dostupné z: doi:10.7146/se.v9i1.118245

MACGREGOR, Alastair. *The sounds of GTA5: Rage Audio Features* [záznam z konference]. Game Developers Conference, 17–21.3.2014, San Francisco, CA, USA. Dostupné z: https://www.youtube.com/watch?v=EUN1j-3IPW0 [cit. 18-6-2023]

MONGEAU, Anne-Sophie. Behind the Sound of No Man's Sky: A Q&A with Paul Weir on Procedural Audio. In: *A Sound Effect*. [online] 2017. [cit. 26-6-2023] Dostupné z: https://www.asoundeffect.com/no-mans-sky-soundprocedural-audio/

NATTIEZ, Jean-Jacques. SAMUELS, Robert. *The Boulez–Cage Correspondence*. Cambridge University Press, 1995. ISBN 0521485584

PLUT, Cale a Philippe PASQUIER. Generative music in video games: State of the art, challenges, and prospects. *Entertainment Computing* [online]. 2020, **33** [cit. 2023-06-30]. ISSN 18759521. Dostupné z: doi:10.1016/j.entcom.2019.100337

PRITCHETT, James. The Music of John Cage. Cambridge University Press, 1993. ISBN 0-521-56544-8

SWEET, Michael. *Writing interactive music for video games: a composer's guide*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0-321-96158-7

WALDER, Collin. *Sounds of Night City: Audio Technology In Cyberpunk 2077* [záznam z konference]. Game Developers Conference, 20–24.3.2023, San Francisco, CA, USA. Dostupné z: https://www.gdcvault.com/play/1029310/Sounds-of-Night-City-Audio [cit. 18-6-2023]

WOOLLER, Rene., BROWN, R. Andrew, et al. *A framework for comparison of processes in algorithmic music systems*. Generative Arts Practice, Sydney, Creativity and Cognition Studios Press, 2005. s. 109–124

Software

Allegro 5 [software]. Open source. Dostupné z: https://liballeg.org/
Asheron's Call II: Fallen Kings [videohra]. 2002. Turbine Entertainment Software. Microsoft Game Studios.
Asteroids [videohra]. 1979. Atari, Inc.
Blender [software]. Open source. Dostupné z: https://www.blender.org/
Call of Duty [videohra]. 2003. Infinity Ward. Activision.
Cervii [videohra]. 1993. Vladimír Chvátil.
Child of Eden [videohra]. 2011. Q Entertainment. Ubisoft.
Civilization V [videohra]. 2010. Firaxis Games. 2K, Aspyr.
Cyberpunk 2077 [videohra]. 2020. CD Projekt Red. CD Projekt.
Fabric [software]. Tazman-Audio. Dostupné z: https://www.tazman-audio.co.uk/fabric20
FMOD [software]. Firelight Technologies. Dostupné z: https://www.fmod.com/
Grand Theft Auto 5 [videohra]. 2013. Rockstar North. Rockstar Games.
Guitar Hero [videohra]. 2005. Harmonix Music System. RedOctane.
Max [software]. Cycling 74'. Dostupné z: https://cycling74.com/products/max
Math.js [software]. Jos de Jong. Dostupné z: https://mathjs.org/
Monkey Island 2: KeChuck's Revenge [videohra]. 1991. LucasArts
No Man's Sky [videohra]. 2016. Hello Games.
No One Lives Forever [videohra]. 2000. Monolith Production. Fox Interactive.
OpenAL [software]. Creative Technology. Dostupné z: https://www.openal.org/
Pure Data [software]. Open source. Originální distribuce: Miller S. Puckette. Dostupné z: https://puredata.info/ [cit. 26-6-2023]
Reactional Music [software]. Reactional Music. Dostupné z: https://reactionalmusic.com/

Red Dead Redemption 2 [videohra]. 2018. Rockstar Studios. Rockstar Games.
Rez [videohra]. 2001. United Game Artists. Sega.
SSEYO Koan Interactive Audio Platform [software]. 1994. SSEYO.
Space Invaders [videohra]. 1978. Taito.
Spore [videohra]. 2008. Maxis. Electronic Arts.
Stellaris [videohra]. 2016. Paradox Development Studio. Paradox Interactive.
Super Mario Bros [videohra].1985. Nintento.
Super Mario Galaxy [videohra]. 2007. Nintendo.
Surge XT [software]. Open source. Dostupné z: https://surge-synthesizer.github.io/
The Elder Scrolls V: Skyrim [videohra]. 2012. Bethesda Game Studios. Bethesda Softworks.
Unity [software]. Unity Technologies. Dostupné z: https://unity.com/
Unreal Engine [software]. Epic Games. Dostupné z: https://www.unrealengine.com/
World of Warcraft [videohra]. 2004. Blizzard Entertainment.
Wwise [software]. Audiokinetic. Dostupné z:
https://www.audiokinetic.com/en/products/wwise/

Příloha 1: HLSL kód pro analýzu obrazu

```
#pragma kernel TagsInit
#pragma kernel TagsMain
#pragma kernel CompleteInit
#pragma kernel CompleteMain

struct tagColor
{
    uint red, green, blue;
};
struct gradientColor
{
    uint red, green, blue;
};
struct tagColorWithCoords
{
    uint red, green, blue, xMax, yMax;
};
struct distance
{
    uint distance;
    tagColor tagColor;
};
struct tagsDistGradient
{
    uint distance;
    tagColor tagColor;
    gradientColor gradientColor;
};
Texture2D<float4> TagText;
Texture2D<float4> DepthText;
Texture2D<float4> GradText;
RWStructuredBuffer<tagColor> TagsOutput;
RWStructuredBuffer<tagColor> TagList;
RWStructuredBuffer<distance> DistanceBuffer;
RWStructuredBuffer<distance> DistanceIn;
RWStructuredBuffer<tagsDistGradient> CompleteOutput;
int TextWidth;
int TextHeight;
uint TagListLength;
int tagCount;
groupshared tagColor TagCache[270];
groupshared tagsDistGradient CompleteCache[270];

//////////TAGS//////////////
[numthreads(64, 1, 1)]
void TagsInit(uint3 groupID : SV_GroupID, // 3D ID of thread group; range depends on Dispatch call
    uint3 groupThreadID : SV_GroupThreadID, // 3D ID of thread in a thread group; range depends on numthreads
    uint groupIndex : SV_GroupIndex, // flattened/linearized SV_GroupThreadID. // groupIndex specifies the index within the group (0 to 63)
    uint3 id : SV_DispatchThreadID) // = SV_GroupID * numthreads + SV_GroupThreadID
{
    TagsOutput[id.x].red = 0;
    TagsOutput[id.x].green = 0;
    TagsOutput[id.x].blue = 0;
}
[numthreads(270, 1, 1)]
void CompleteInit(uint3 id : SV_DispatchThreadID)
{
    CompleteOutput[id.x].distance = 0;
    CompleteOutput[id.x].tagColor.red = 0;
    CompleteOutput[id.x].tagColor.green = 0;
    CompleteOutput[id.x].tagColor.blue = 0;
    CompleteOutput[id.x].gradientColor.red = 0;
    CompleteOutput[id.x].gradientColor.green = 0;
    CompleteOutput[id.x].gradientColor.blue = 0;
}
[numthreads(270, 1, 1)]
void TagsMain(uint3 groupID : SV_GroupID, // 3D ID of thread group; range depends on Dispatch call
    uint3 groupThreadID : SV_GroupThreadID, // 3D ID of thread in a thread group; range depends on numthreads
    uint groupIndex : SV_GroupIndex, // flattened/linearized SV_GroupThreadID. // groupIndex specifies the index within the group (0 to 63)
    uint3 id : SV_DispatchThreadID) // = SV_GroupID * numthreads + SV_GroupThreadID
{
    int check;
    int column;
    TagCache[groupIndex].red = 0;
    TagCache[groupIndex].green = 0;
    TagCache[groupIndex].blue = 0;
    GroupMemoryBarrierWithGroupSync();
```

```
    for (column = 0; column < TextWidth; column++)
    {
        uint3 col = uint3(255.0 * TagText[uint2(column, groupIndex)].rgb);
        if (!(col.r == 0 && col.g == 0 && col.b == 0))
        {
            for (check = 0; check < 270; check++)
            {
                if (TagCache[check].red == col.r && TagCache[check].green == col.g && TagCache[check].blue == col.b)
                {
                    break;
                }
                if (check == 269)
                {
                    TagCache[groupIndex].red = col.r;
                    TagCache[groupIndex].green = col.g;
                    TagCache[groupIndex].blue = col.b;
                }
            }
        }
    }
    GroupMemoryBarrierWithGroupSync();
    for (int j = 0; j < 270; j++)
    {
        TagsOutput[j + (270 * groupID.x)].red = TagCache[j].red;
        TagsOutput[j + (270 * groupID.x)].green = TagCache[j].green;
        TagsOutput[j + (270 * groupID.x)].blue = TagCache[j].blue;
    }
}
/////GRADIENT_AND_DISTANCE/////
[numthreads(270, 1, 1)]
void CompleteMain(uint3 groupID : SV_GroupID, // 3D ID of thread group; range depends on Dispatch call
    uint3 groupThreadID : SV_GroupThreadID, // 3D ID of thread in a thread group; range depends on numthreads
    uint groupIndex : SV_GroupIndex, // flattened/linearized SV_GroupThreadID. // groupIndex specifies the index within the group (0 to 63)
    uint3 id : SV_DispatchThreadID) // = SV_GroupID * numthreads + SV_GroupThreadID
{
    int column;
    CompleteCache[groupIndex].tagColor.red = TagList[groupID.x].red;
    CompleteCache[groupIndex].tagColor.green = TagList[groupID.x].green;
    CompleteCache[groupIndex].tagColor.blue = TagList[groupID.x].blue;
    CompleteCache[groupIndex].distance = 0;
    CompleteCache[groupIndex].gradientColor.red = 0;
    CompleteCache[groupIndex].gradientColor.green = 0;
    CompleteCache[groupIndex].gradientColor.blue = 0;
    for (column = 0; column < TextWidth; column++)
    {
        uint3 col = uint3(255.0 * TagText[uint2(column, groupIndex)].rgb);
        uint3 grad = uint3(255.0 * GradText[uint2(column, groupIndex)].rgb);
        uint dist = uint(255.0 * DepthText[uint2(column, groupIndex)].r);
        if (TagList[groupID.x].red == col.r && TagList[groupID.x].green == col.g && TagList[groupID.x].blue == col.b)
        {
            if (dist > CompleteCache[groupIndex].distance)
            {
                CompleteCache[groupIndex].distance = dist;
                CompleteCache[groupIndex].gradientColor.red = grad.r;
                CompleteCache[groupIndex].gradientColor.green = grad.g;
                CompleteCache[groupIndex].gradientColor.blue = grad.b;
            }
        }
    }
    GroupMemoryBarrierWithGroupSync();
    uint row;
    uint rowMax = 0;
    for (row = 1; row < 270; row++)
    {
        if (CompleteCache[row].distance > CompleteCache[rowMax].distance)
        {
            rowMax = row;
        }
        CompleteOutput[groupID.x].tagColor.red = CompleteCache[rowMax].tagColor.red;
        CompleteOutput[groupID.x].tagColor.green = CompleteCache[rowMax].tagColor.green;
        CompleteOutput[groupID.x].tagColor.blue = CompleteCache[rowMax].tagColor.blue;
        CompleteOutput[groupID.x].distance = CompleteCache[rowMax].distance;
        CompleteOutput[groupID.x].gradientColor.red = CompleteCache[rowMax].gradientColor.red;
        CompleteOutput[groupID.x].gradientColor.green = CompleteCache[rowMax].gradientColor.green;
        CompleteOutput[groupID.x].gradientColor.blue = CompleteCache[rowMax].gradientColor.blue;
    }
}
```