Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Det humanistiske fakultet
Institutt for musikk

Sigmund K. Bakheim

Algokomp

En utforsking av programmering som kreativ prosess med tilfeldighet som materiale

Bacheloroppgave i Musikkteknologi
Veileder: Øyvind Brandtsegg

Mai 2021

NTNU
Kunnskap for en bedre verden

Sigmund K. Bakheim

Algokomp

En utforsking av programmering som kreativ prosess med tilfeldighet som materiale

Bacheloroppgave i Musikkteknologi
Veileder: Øyvind Brandtsegg
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Det humanistiske fakultet
Institutt for musikk

NTNU

Kunnskap for en bedre verden

Sammendrag

Med denne rapporten ønsker jeg å beskrive teknisk utvikling og kreativ bruk av dataprogrammet Algokomp, som genererer komposisjoner automatisk gjennom tilfeldig utvalg av forhåndsbestemt lydmateriale. Arbeidet har i stor grad vært preget av en praktisk og utforskende fremgangsmåte, noe som vil gjenspeiles i rapporten. Arbeidet har bidratt til en økt forståelse for programmering som kreativ prosess, bruk av tilfeldighet i musisering, og datamaskinen som kunstnerisk samarbeidspartner. I rapporten vil jeg reflektere rundt og diskutere disse temaene i lys av mitt eget arbeid.

Abstract

The contents of this paper describe the technical development and creative usage of the computer program Algocomp, which generates compositions automatically through random arrangement of previously decided sound material. The working process has been characterized by a largely practical and exploratory nature, which will be reflected in this report. The project has led me to an increased understanding for subjects such as programming as a creative process, usage of randomness in music, and the computer as an artistic companion. In this paper I will reflect on these subjects in the light of my own work.

Innholdsfortegnelse

1 Introduksjon og motivasjon

Jeg ble inspirert til å utvikle Algokomp i løpet av våren 2020, gjennom faget MUST2062 Digital komposisjon og framføring, hvor vi hver uke jobbet utforskende med digitale musiseringsverktøy i form av csound-patcher. Da vi hadde algoritmisk komposisjon som tema, fikk jeg blant annet testet ut Øyvind Brandtseggs Sound_chart.csd, som tar utgangspunkt i John Cages' I Ching. I Ching er en en såkalt aleatorisk arbeidsmetode hvor eldgamle kinesiske spådomspinner med markeringer ble kastet på bakken og lest av for å avgjøre forskjellige egenskaper ved musikken. Sound_chart.csd kan sies å være en digitalisering av denne arbeidsmetoden. Patchen brukes i kombinasjon med 12 samples av forskjellige orkesterlyder og komponerer lydforløp ved å spille av tilfeldige samples etter en forhåndsbestemt rytme/tid. Jeg ble fort interessert i å teste ut programmet med egne lydfiler som materiale for komponering. En tidlig idé var å anvende en gruppe samples hvor alle lyder holdt seg til én bestemt toneart. Jeg så for meg at musikken da ville oppleves koherent men uforutsigbar på samme tid. Jeg ønsket også å la tidspunkt for trigging av samples tilfeldig.

Jeg utviklet en tidlig versjon av programmet i løpet av vårsemesteret 2020, som med over hundre samples, arrangerte komposisjoner med både tilfeldig avgjort valg av sample og plassering i tid. Jeg opplevde prosessen rundt utviklingen av programmet som svært engasjerende. Det var utrolig spennende å generere musikk på denne måten fordi jeg følte at jeg enkelt kunne generere tilfredsstillende komposisjoner uten å gjøre stort mer enn å trykke på «compile»-knappen og vente på resultatet. For meg ble utviklingen av programmet Algokomp nøkkelen til å forstå musikk på en ny måte. Musiseringen handlet ikke lenger om å velge hvilken note som skulle plasseres hvor i komposisjonen, det var programmets oppgave. Min jobb var å utvikle alle forutsetningene for programmets virkemåte via programmering og valg av samples. Interessen min for programmering som kreativt verktøy ble forsterket, og jeg begynte å stille meg selv noen spørsmål knyttet til mitt eget arbeid, datamaskinens rolle som kunstnerisk samarbeidspartner og hvorfor tilfeldighet var så engasjerende å jobbe med. Jeg presenterte den tidlige versjonen av Algokomp under eksamen i MUST2062, og kort tid etter bestemte jeg meg for å videreutvikle programmet til min bacheloroppgave i musikkteknologi.

2 Metode

Arbeidets metode omfatter både den tekniske utviklingen av Algokomp så vel som kreativt arbeid med komposisjon gjennom programmet. Selv om de to aspektene et knyttet tett sammen, tror jeg det er nyttig å gjøre rede for disse hver for seg, da jeg må bruke hodet på svært forskjellige måter avhengig av del av prosessen jeg jobber med til enhver tid. Den tekniske utviklingen av koden er preget av problemløsning og ryddighet, og det finnes metoder som er objektivt bedre enn andre. På den andre siden er det kreative arbeidet rundt samplevalg og anvendelse av programmet mer avhengig av subjektive preferanser, og hvorvidt ett valg er bedre enn et annet er avhengig av enhvers smak.

2.1 Teknisk utvikling av Algokomp

Kort fortalt bestod den tekniske utfordringen min av å utvikle en csound-patch som kunne generere komposisjoner via tilfeldig utvalg av samples, og tilfeldig plassering av disse innen et gitt tidsrom. Første bud var å få implementert alle samples inn i Csound.

konvertering fra lydfil til tabell

En enkel måte å få implementert lydfiler inn i Csound, er via programmets innebygde GEN01-rutine via ftgen-opkoden, som analyserer en gitt lydfil, konverterer informasjonen, og lagrer den i en funksjonstabell med en unik tallverdi. Denne metoden var brukt i den tidligere nevnte Sound Chart-patchen, som Algokomp ble utviklet på grunnlag av. Et eksempel på konvertering av flere lydfiler via GEN01-rutinen i Csound kan se slik ut:

> 12 samples av gitar og 16 samples av piano konverteres via Csounds innebygde ftgen-opkode ved å oppgi navn + filsti på de filene man ønsker å konvertere. Etter konvertering sitter man igjen med tabellene «ftable 1 – 28». Man kan dermed spille av lyd fra disse tabellene ved hjelp av loscil-opkoden.

Fordelen ved bruk av funksjonstabeller er at programmet enkelt kan referere til og anvende disse via deres tallverdier. Å forholde seg til heltall for å kalle på samples er mer praktisk og ryddig enn å være nødt til å oppgi hvert enkelt samples navn og filsti.

Problemer ved Algokomps tidligste fase

I Algokomps tidligste fase brukte jeg ftgen-opkoden for å konvertere all lyd med en egen linje kode for hvert sample. I starten var dette greit, men ettersom antallet lydfiler økte, forstod jeg at jeg måtte finne en bedre løsning. Jeg hadde over hundre samples, og planer om å implementere mange fler, og storparten av koden min bestod av konvertering av filer til tabell via ftgen-opkoden. I tillegg til at det var tungvindt å skrive inn hvert sample manuelt, ble også hvert sample knyttet til en fast tabellverdi, og om jeg skulle ønske å fjerne ett eller flere samples fra programmet, kunne dette bety tomrom i tabellenes tallrekke, og feilmeldinger om loscil-opkoden forsøkte å kalle på en tabell som ikke eksisterte.

Programmet var forøvrig fungerende, og klarte å generere tilfeldige komposisjoner, men jeg forstod at om jeg skulle videreutvikle Algokomp, var jeg nødt til å finne en ryddigere løsning. Det var faktisk svært mange aspekter ved koden jeg ikke forstod, noe som kan skyldes at jeg tok utgangspunkt i en patch jeg ikke hadde utviklet selv. Et lyspunkt i det hele var at jeg hadde klart å realisere idéen min, og etter litt refleksjon forstod jeg hvilke deler av koden som var viktigst, og hva jeg godt kunne kvitte meg med. Derfor begynte jeg i Januar 2021 å utvikle programmet på nytt, med blanke ark og en bedre forståelse for hva arbeidet mitt innebar.

Optimalisering av samplekonvertering

Jeg ønsket å få implementert lydfilene mine inn i Csound, men på en ryddigere og mer fleksibel måte Jeg så for meg at ved å la Csound lete gjennom samplemapper, for så å utføre konvertering av disse mappenes innhold til tabeller, ville jeg enkelt kunne endre på mappenes innhold, uten at dette ble et problem ved kjøring av programmet. For å løse denne utfordringen brukte jeg en kombinasjon av arrays, directory-opkoden, until-løkker og en UDO av Joachim Heintz. Det var nokså utfordrende å få dette til å fungere, da jeg hadde lite erfaring med disse teknikkene på forhånd, men jeg vil si at resultatet ble en elegant løsning, og langt bedre enn Algokomps tidligste versjon.

Mappestrukturen

I og med at denne løsningen baserer seg på gjennomgang av samplemappene for å implementere lydfiler krever den at jeg bruker en fast mappestruktur, hvor hver samplemappe inneholder tre submapper som jeg har valgt å kalle forgrunn, middelgrunn og bakgrunn, eller BG, MG og FG. I disse tre mappene ligger det mapper med lydfiler i, som programmet kan hente inn i Csound og konvertere til tabell. Det å forholde seg til en rigid

mappestruktur er en begrensning, men det er en begrensning som tillater mye fleksibilitet. En stor fordel ved denne metoden er at den tillater sortering av lydlig materiale, noe jeg vil snakke mer om senere i rapporten, men først vil jeg gjøre rede for hvordan metoden fungerer rent teknisk.

Directory, Arrays og Fleksibilitet

I samme mappe hvor jeg har lagret prosjektfilen min, Algokomp.csd, ligger også alt lydmateriale sortert i samplemapper. I programmet kan jeg oppgi et tall som bestemmer hvilken av samplemappene Algokomp skal benytte seg av for komposisjonen. Ved oppstart av programmet brukes directory-opkoden for å gå lete gjennom den valgte samplemappens BG-, MG- og FG-mapper. For hver av disse brukes igjen directory-opkoden for å lagre alle mapper i arrays. Arrays minner litt om tabeller, da de de lagrer informasjon, som man enkelt kan aksessere senere. Forskjellen i dette tilfellet er at jeg bruker arrays for å lese av og lagre mappenavnene fra den tidligere valgte samplemappen som strenge-variabler, og senere bruke disse variablene for å konvertere lydfilene til tabell. Dette betyr blant annet at jeg kan anvende nye samplemapper eller endre på mappenes innhold uten å være nødt til å skrive en eneste linje kode.

Tilfeldig utvalg av instrumentmapper

Etter at navnene på instrumentmappene fra BG-, MG- og FG-mappene har blitt implementert som strenger i arrays, brukes opkoden ftsamplebank, som konverterer alle filer i en mappe til tabell. Men før lydfilene konverteres, må programmet avgjøre hvor mange, og hvilke mapper som skal brukes ved komposisjon. Dette gjøres tilfeldig via en UDO av Joachim Heinz, som tillater tilfeldig utvalg av elementer fra arrays. En annen operasjon fra samme UDO brukes deretter for å fjerne valgte mapper fra utvalget, slik at de ikke velges to ganger ved samme komposisjon.

Etter tilfeldig utvalg av mapper konverteres alle lydfiler til tabell. På dette stadiet i prosessen har jeg sørget for å legge til litt kode som holder styr på hvor mange samples programmet har hentet inn fra BG, MG og FG-mappene. Dette gjøres simpelthen ved hjelp av tellere og egendefinerte variabler. Dette har jeg gjort for å sikre meg økt kontroll i komposisjonsprosessens neste fase, nemlig tilfeldig utvalg og avspilling av samples.

Tilfeldig arrangement av tilfeldige samples

Med konverteringen av lydfiler til tabeller på en oversiktlig måte er det vanskeligste unnagjort, og selve arrangeringen utføres på en veldig simpel måte, gjennom det jeg kaller sampletriggende instrumenter. Jeg har mange slike instrumenter i koden, men det er som regel kun en eller to instanser av disse som tas i bruk under hver komposisjon. Så godt som all kreativ og utforskende bruk av koding i de senere fasene av prosjektet har foregått i denne delen av koden som avgjør de formmessige aspektene ved komposisjonene, som plassering av samples i tid og antall avspilte samples under en bestemt komposisjon. Her er et eksempel på et sampletriggende instrument:

```
268 instr 10
269 iRand random p4, p5
270 iRound = round(iRand)
271 ilen = ftlen(iRound)
272 iDur = ilen/sr
273 iDelay random 0, 80
274
275         iCount = 0
276     until iCount >= 24 do
277         event_i "i", 100, iDelay, iDur, iRound
278         iRand random p4, p5
279         iDelay random 0, 80
280         iRound = round(iRand)
281         ilen = ftlen(iRound)
282 ;       print iDur
283 ;       print iDelay
284         iDur = ilen/sr
285         iCount += 1
286     enduntil
287 endin
```

I øverste bolk av koden ser vi at instrumentet tar inn verdiene p4 og p5, velger et tilfeldig tall innen spennet mellom disse verdiene, og runder av til heltall. Dette tallet representerer hvilket tabellnummer som skal spilles av under komposisjonen p4 og p5 er hentet fra tellervariablene som holder oversikt over antall implementerte samples fra hver kategori nevnt tidligere i rapporten. I dette tilfellet omfatter p4 og p5 alle verdier fra BG-mappen, og sampletriggeren vil kun trigge lyder derfra. I tillegg kalkulerer koden lengden på hvert tilfeldig

valgte sample, og bruker dette som varighet for avspilling i instrument 100, hvor loscil-opkoden spiller lyd fra tabell via et master-instrument.
Nedre bolk av koden benytter seg av en until-løkke som sørger for at all koden gjentas helt til variabelen iCount er lik eller høyere enn 24. variabelen iDelay velger tilfeldige verdier mellom 0 og 80, som representerer tidspunkt for avspilling av samples oppgitt i sekunder. Legg merke til at de 5 øverste linjene gjentas senere i instrumentet. Dette er fordi verdiene i variablene som skal sendes til instrument 100 trenger å defineres. Disse variablene genererer nye verdier som overskriver hverandre så lenge instrumentet kjører.

Flere sampletriggere og Algokomps kreative feedback-loop

Alle sampletriggende instrumenter har tydelige likheter med eksempelet ovenfor, men med små justeringer. Det de har til felles er at de via en event-opkode får et annet instrument til å spille av samples fra tabell. Noen av instrumentene trigger mange samples i klynger innen et lite tidsrom. Dette er en måte å få programmet til å generere noe som minner om akkorder. En annen instans forsøker å unngå at samples fra én bestemt kategori overlapper, som er nyttig om man for eksempel skal bruke samples av tale. Svært ofte er de tekniske valgene inspirert av at jeg lytter til programmets egne komposisjoner, for så å forsøke å gjenskape ønskede aspekter ved musikken, eller å korrigere det jeg ikke føler passer inn. For meg er dette noe av det mest interessante ved å komponere i Algokomp, og et tema jeg vil komme tilbake til. Men først vil jeg ta for meg de kunstneriske aspektene og valgene ved programmets virkemåte.

2.2 Kreativ bruk av Algokomp

Selv om Algokomp er designet for å generere lydforløp automatisk, er det viktig å påpeke at det ikke er datamaskinen som gjør all jobben. Arbeidsoppgavene er fordelt mellom meg og programmet, og min jobb dreier seg rundt å tilrettelegge for at den tilfeldige arrangeringen kan bli til god kunst. Med andre ord er produksjon av musikk i Algokomp preget av uforutsigbarhet og begrenset kontroll. Å forholde seg til dette betyr at man må tenke fundamentalt annerledes på komposisjonsprosessen, enn hva man gjør ved produksjon i DAWs (Digital Audio Workstation), som Logic, Ableton og FL-studio, eller ved tradisjonell noteringskomposisjon på noteark eller i programmer som Sibelius.

Hva kan jeg bestemme?

Når jeg kjører programmet og Algokomp begynner å generere en komposisjon, har jeg null kontroll over programmets valg. Derfor er det kun gjennom forarbeidet at jeg kan påvirke komposisjonene. Forarbeidet dreier seg i korte trekk rundt:

- Lydmateriale i form av samples, og hvordan disse er sortert i mappestrukturen
- Valg av aktiv samplemappe under komposisjonen
- Valg av antall aktive instrumentmapper fra mappene BG, MG og FG
- Oppsett av sampletriggende instrumenter, hvilke tabeller de skal spille fra, og hvordan lydene spilles av
- Valg av impulsrespons for innebygd konvolusjonsklang-effekt
- Lengde på komposisjonen
- Seed for komposisjonen

Disse aspektene kan jeg kontrollere ved å justere på koden i programmet, eller ved å gjøre lydopptak eller på andre måter skaffe lydmateriale, som jeg plasserer i en kompatibel mappestruktur. Når jeg gjør slike valg definerer jeg rammeverket for Algokomps muligheter for tilfeldige valg under komposisjon, så selv om jeg ikke kan bestemme nøyaktig hvilke samples som plasseres hvor i lydbildet, har jeg allikevel mye makt over hvor tilfeldige tilfeldighetene er. Her er en oppsummering av mine kreative valg og deres påvirkning av en vanlig Algokomp-komposisjon:

Eksempel på bruk av Algokomp

Jeg fyller en mappestruktur med samples som jeg ser for meg vil passe godt sammen, og pleier derfor å forholde meg til tonale samples innen én bestemt toneart, i tillegg til samples som ikke har noen bestemt tonalitet, som ambiens eller tale. Jeg plasserer alle instrumentmappene med samples i mappene Background, Middleground og Foreground. Atmosfæriske samples plasserer jeg som regel i bakgrunnsmappen, mens melodier og toner plasseres i middel- og forgrunnsmappene. Det er selvsagt opp til en selv hvordan man ønsker å sortere instrumentmappene sine.

Deretter må jeg fortelle programmet at det skal anvende denne spesifikke samplemappen. På grunn av dette er det er viktig at samplemappen har navnet «Samples» etterfulgt av et tall. Hvilken samplemappe man anvender

til enhver komposisjon avgjøres med variabelen «ifn» i linje 23. Deretter velger man hvor mange instrumentmapper man skal bruke fra hver av de tre mappene BG, MG og FG i linjene 55-57. Dette kan enten velges manuelt eller tilfeldig. Selv liker jeg å teste ut forskjellige kombinasjoner ved å generere flere komposisjoner etter hverandre med forskjellig «Seed», en variabel i linje 14 som endrer på programmets valg under komposisjon. Om man tror man har opprettet et godt rammeverk for programmets forutsetninger, er dette den beste måten å oppnå ulike resultater på uten å måtte gjøre mange ytterlige endringer.

Ved justering av de sampletriggende instrumentene kan man prege de formmessige og tidsrelaterte aspektene ved komposisjonen, i tillegg til å bestemme hvilke samples utvalget skal gjøres fra, og hvilket lydavspillende instrument som skal anvendes. Enten instrument 100 for tørt signal, eller 101 for konvolusjonsklang.

Konvolusjonsklang

Den eneste signalprosesseringseffekten jeg har inkludert i programmet er en konvolusjonsklang via Csounds innebygde cvanal-opkode, som lar meg konvertere lydfiler fra WAV til cva, og bruke disse som impulsrespons. Jeg velger å skrive om dette under den kreative delen av arbeidet, fordi det er opp til meg å velge impulsrespons for hver komposisjon. Effekten fungerer godt i samspill med programmet, og bidrar til å skape et mer interessant lydbilde. Konvolusjonsklang har et stort spenn på hvor mye den preger de påvirkede lydene, så noen ganger vil resultatet bli nokså subtilt, og andre ganger svært merkbart. Jeg har lagt ved noen komposisjoner fra Algokomp, hvor to av disse gir en god illustrasjon av konvolusjonsklangens effekt på musikken.

I gitareksempelet har jeg brukt lyden av en teskje i en kopp som impulsrespons. Resultatet er at gitarlyden får forsterket de metalliske, skarpe frekvensene med mye aktivitet i området 4KHz til 20KHz, i tillegg til en ekko-effekt som følge av at impulsresponsen inneholder flere tydelige transienter. Lyden gir meg assosiasjoner til et østlig strengeinstrument, noe jeg føler bidrar til å berike atmosfæren og personligheten til komposisjonen på en nesten historiefortellende måte.

I det andre eksempelet har jeg brukt et taleklipp som impulsrespons. I motsetning til teskje-samplet bærer denne lyden et mykere preg, uten harde transienter og mer behagelige frekvenser fra 100Hz til 3KHz. I den vedlagte komposisjonen er det lett å høre at konvolusjonsklangen tar er mer anonym rolle i lydbildet, og ligger mer som et teppe i bakgrunnen. Taleklippet er også lavere i volum, og varer litt over 3 sekunder, som gjør det

over dobbelt så langt som teskje-samplet og fører til at klanghalen blir betydelig lenger. Kreativ bruk av konvolusjonsklang er et svært nyttig virkemiddel som kan brukes til å oppnå svært varierende resultater.

3 Kontekstualisering

Algokomp er designet rundt tilfeldighet, dermed kan man trygt si at Algokomp faller innenfor kategorien aleatorisk musikk som Store Norske Leksikon definerer som «musikk hvor det eksisterer uforutsigbare elementer bestemt av tilfeldighet via f.eks. Terningkast.» (Holter, 2020) Ved å sammenligne Algokomp med andre verker innen aleatorisk musikk ønsker jeg å kontekstualisere programmet og peke ut noen avgjørende kjennetegn ved bruk av tilfeldighet under musisering.

Aleatorisk musikk

Ordet Aleatorikk kommer fra det latinske «alea» som betyr terning. Det sies at Mozart skal ha benyttet seg av terninger for å arrangere valser allerede på 1790-tallet. På 1860-tallet utviklet amerikaneren John Clinton «The Quadrille Melodist», et sett med korte musikalske fraser på noteark som var designet for å kunne settes sammen i hvilken som helst rekkefølge for å skape over 428 millioner unike komposisjoner. Som den britiske komponisten David Bruce påpeker i en youtube-video om algoritmisk komposisjon var dette mulig takket være det faktum at Quadrille-sjangeren var nokså simpel, og derfor kunne dekonstrueres ned til et sett med regler. (Bruce, 2020). Algokomp bruker i likhet med kvadriljemelodisten til John Clinton, tilfeldig arrangement av et forhåndsbestemt materiale for å lappe sammen et enormt antall potensielle komposisjoner. Da jeg begynte å utvikle Algokomp valgte jeg å la programmet være fritt fra noen form for rytme og tempo, noe som er et kjennetegn av musikk fra ambient-sjangeren. Kanskje er Algokomp litt som John Clintons kvadriljemelodist, bare rettet mot en annen sjanger?

Terry Rileys «In C» er en komposisjon designet for å kunne fremføres på utallig mange forskjellige måter, et tema som åpenbart går igjen i musikk preget av tilfeldighet. Det interessante med komposisjonen er at den legger opp til at de individuelle medlemmene av orkesteret selv får bestemme om de skal bevege seg videre komposisjonen, eller repetere samme frase om og om igjen, noe som fører til at det er umulig å forutse hvordan en framføring vil høres ut. Takket være at musikken er skrevet med dette i tankene, kan musikken være både kaotisk og koherent på samme tid. Her finnes det også klare likheter med Algokomp. Jeg er jo alltid klar

over at lydmaterialet jeg velger skal randomiseres, noe som preger mitt utvalg av samples jeg gir til programmet. Som David Bruce nevner i en annen video er kanskje det mest interessante ved «In C» at det inneholder mange gradvise endringer, knyttet til forskjellige musikalske konsepter som tonelengde, takt, skala og instrumentasjon. (Bruce, 2019) Det aleatoriske elementet sørger for at disse interessante teksturene som oppstår når repetisjonene overlapper hverandre, aldri er helt like, noe som gjør at «In C» forblir et spennende stykke musikk.

Aleatorikk blir ikke kun brukt for å skape klassiske stykker eller for å utfordre kunstneriske konvensjoner, men er en svært fleksibel teknikk, som kan brukes slik en selv ønsker. I John Cages bruk av aleatorikk i «I Ching» var han til tider ekstremt spesifikk da han skulle velge hvilke aspekter som skulle avgjøres tilfeldig (Baofu 2012) Et eksempel på bruk av aleatorikk innen svært snevre rammer. Fra pop-verdenen kan man dra fram Brian Enos bruk av aleatorikk for å produsere braksuksessene «Joshua Tree» og «Viva La Vida» med U2 og Coldplay via en egen oppfinnelse inspirert av John Cages «I Ching», et sett med instruksjonskort kalt «Oblique Strategies». (Inspirational Working Methods, 2017) Dette minner på mange måter om å dra lapper ut fra en hatt for å avgjøre kreative valg ved komposisjon, som jeg selv har opplevd er en svært nyttig ting å prøve når man opplever kreativ skrivesperre.

I SNLs artikkel om aleatorikk nevnes det at improvisasjon innen gitte rammer er et viktig aspekt ved metoden. (Holter, 2020) Jeg tolker dette som at restriksjoner er viktige for den aleatoriske prosessen, noe jeg har fått erfare gjennom utviklingen av Algokomp. Når jeg produserer musikk i DAW sliter jeg ofte med å få idéer, og det kan være vanskelig å løsrive seg fra å bruke de samme effektene og instrumentene man alltid bruker av vane. I tillegg er moderne DAWs på mange måter designet rundt takt og tempo, som var irrelevante for musikken jeg i dette tilfellet ønsket å lage. Det jeg opplevde under utviklingen av Algokomp var at restriksjonene mine rundt valg av samples og plassering av disse under komposisjon tvang meg til å tenke nytt.

Det er ikke kun i musikken at tilfeldighet kan være en kilde til god kunst. I et intervju av Adresseavisen med Håkon Bleken fra 2019 maler trondheimskunstneren et collage-maleri mens han svarer på spørsmål om prosessen rundt å skape et slikt kunstverk. Han påpeker, slik han også har gjort i et annet nylig intervju med Trønder-TV, at tilfeldighet og flaks ofte er avgjørende for hvordan kunstverkene hans ender opp. Blekens collage-malerier er ofte malt på grunnlag av et lerret dekket med papirutklipp i forskjellige fasonger fra aviser, bøker og ukeblader. Disse er klistret til lerretet på en nokså kaotisk og tilfeldig måte, og noen ganger er disse

satt sammen av noen andre enn Bleken selv, slik at tilfeldighetselementet er sterkere, og minst mulig preget av kunstnerens egne preferanser. Når han begynner å male ser han etter former i utklippene, noe gjenkjennbart som kan bli til noe i maleriet. Han understreker at det til enhver tid er viktig å ha et åpent sinn rundt maleriets utvikling, slik at man ikke låser seg til en bestemt idé for tidlig. (Adresseavisen, 2019) Jeg kan kjenne meg igjen i den kunstneriske prosessen han beskriver på flere måter, med tanke på at jeg har gjort valg inspirert av programmets egne komposisjoner. Det kunne ha vært interessant å latt noen andre bruke Algokomp med egne samples, og sett om de utviklet programmet i en annen retning.

4 Refleksjon

Gjennom arbeidet med Algokomp har jeg lært mye, både om koding og generell utvikling av digitale komposisjonsverktøy, og om de forskjellige aspektene ved aleatorisk musikk. For å gi en god oppsummering av mine egne resultater, ønsker jeg å ta et objektivt blikk på programmets styrker og svakheter, og diskutere hva jeg kunne ha gjort annerledes.

Koding

I løpet av arbeidet med programmeringen oppstod det stadig vekk små problemer som jeg har latt være å fokusere på i rapporten, men jeg føler det er viktig å anerkjenne dette, da det utvilsomt har preget tidsbruken under utviklingen av programmet. Mangel på erfaring med koding kan føre til at man må bruke mye tid på noen få problemer, eller ta til takke med en alternativ løsning. Derfor har jeg lært at det er viktig å be om hjelp når man står fast, og å vite hvor man kan finne svar på problemer som oppstår underveis. Jeg er stort sett fornøyd med programmets kode, men innser at noen ting kan forbedres. Som min veileder, Øyvind Brandtsegg har poengtert er det ikke nødvendig å utføre konvertering av lydfil til .cva-fil for impulsrespons i selve Algokomp-patchen, da dette kan oppleves som forvirrende. I tillegg ble det nevnt at jeg kan bestemme lengden på komposisjonene via de sampletriggende instrumentene, i stedet for å bruke en statisk verdi definert nederst i programmets kode. Dette var noe jeg ikke hadde tenkt over at var nyttig, men så brått at det er noe jeg burde få implementert i programmet.

Endring av planer underveis

I utgangspunktet skulle ikke Algokomp være et komposisjonsverktøy, men heller en fullstendig autonom komponist, som brukte et stort utvalg av algoritmer som f.eks markovkjeder og cellular automata for å arrangere lydmaterialet. Jeg fant i stedet ut at evnen til å justere på noen av parameterne i programmet gjorde komposisjonsprosessen langt mer engasjerende. Jeg hadde også i utgangspunktet tenkt at et stort utvalg av signalprosesseringseffekter ville bidra til å gi mer dybde og variasjon til komposisjonene, men jeg kom ikke så langt. Jeg var for det meste opptatt av å utvikle kjerneidéen, og ikke overkomplisere programmet. Jeg synes veien videre for Algokomp ser lys og åpen ut, og det er finnes mange musikalske idéer man kan utforske aleatorisk gjennom utvikling av programmet, noe jeg ønsker å gjøre.

Oppsummering

Arbeidet med utviklingen av det aleatoriske komposisjonsverktøyet Algokomp har vært en lærerik og verdifull prosess for min del. Jeg har lært mye om kreativ programmering i Csound og bruk av aleatorikk i musikkproduksjon. Jeg er stolt over å kunne si at jeg har utviklet et eget program som legger opp til en alternativ måte å skape musikk på.

5 Kilder

Holter, Stig Wernø. 2020 *Aleatorikk*. Hentet 20. Mai 2021.
https://snl.no/aleatorikk

Bruce, David. 2020 *How to Compose Music With a Recipe (Arvo Pärt, Xenakis and others)*. Hentet 20. Mai 2021
https://www.youtube.com/watch?v=X0-zuS_62wc&t

Bruce, David. 2019 *Terry Riley's «In C» - Much more than Minimalism*
https://www.youtube.com/watch?v=JN0bW3ilqF4

Baofu, Peter. 2012 *The Future of Post-Human Performing Arts: A Preface to a New Theory of the Body and its Presence*, Unabridged edition, Cambridge Scholars Publishing s.245

Inspirational Working Methods, 2017 *Inspirational Working Methods: John Cage and the I Ching* Hentet 20. Mai 2021
https://www.youtube.com/watch?v=uyjOnqzjqpc

Adresseavisen. 2019 *Slik lagde 91-åringen sitt aller nyeste kunstverk* Hentet 20. Mai 2021
https://www.adressa.no/pluss/kultur/2020/11/21/Slik-lagde-91-%C3%A5ringen-sitt-aller-nyeste-kunstverk-23030617.ece

6 Vedlegg

1. Komposisjoner – Inneholder 4 eksempler på musikk generert i Algokomp. Disse eksemplene demonstrerer:

   Komposisjon med AB-form, hvor instrument endres halvveis gjennom komposisjonen.

   Komposisjon hvor samples trigges i klynger

   En komposisjon med gitar og konvolusjonsklang

   En komposisjon med softsynth og konvolusjonsklang

2. Samples1 – Inneholder et sett med samples sortert i korrekt mappestruktur.

3. Samples2 – Et sett med andre samples sortert i korrekt mappestruktur.

4. Samples3 – Enda et sett med samples sortert i korrekt mappestruktur.

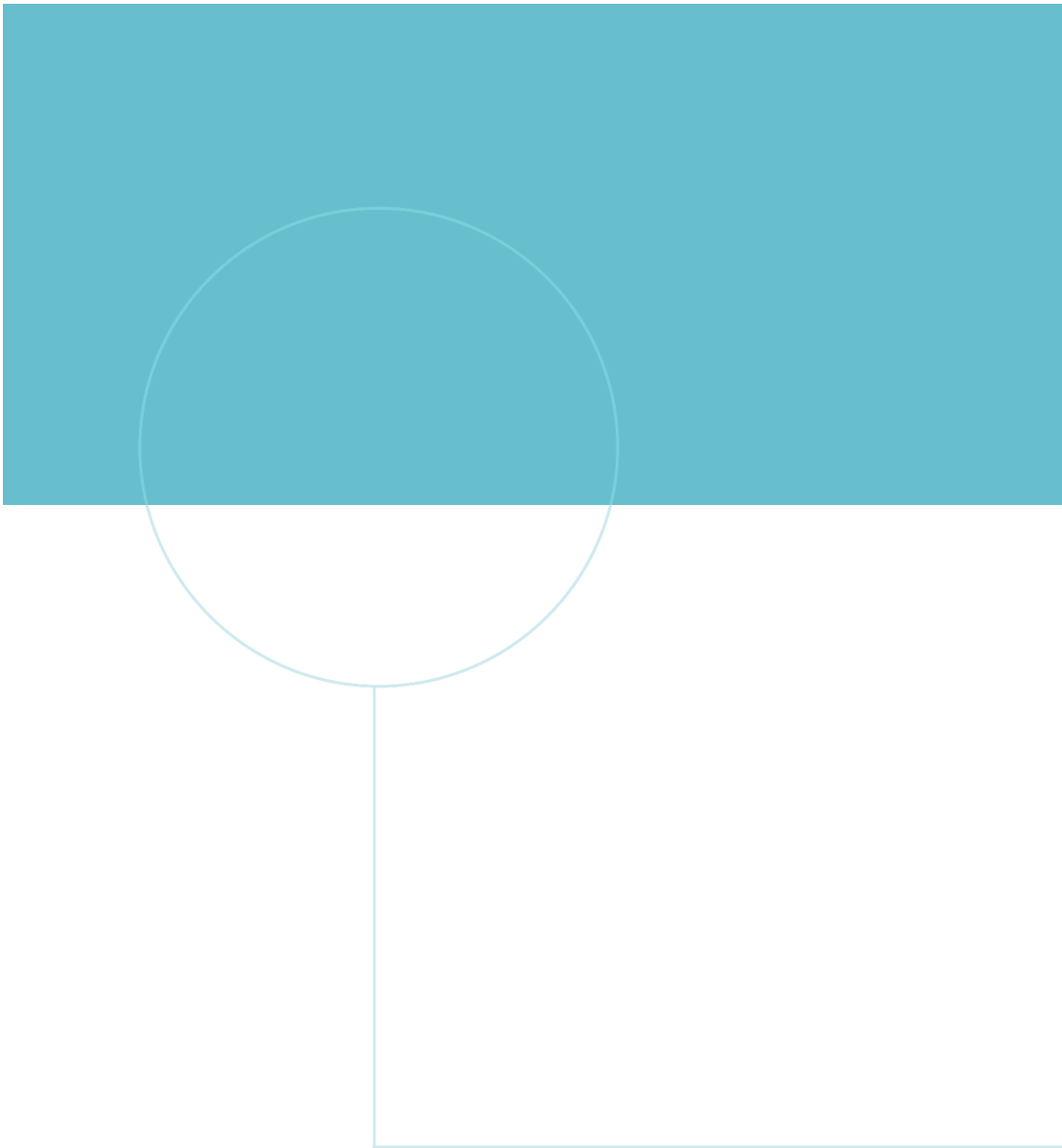
5. Algokomp.csd – Selve programmet

6. Arrays.csd – UDO av Joachim Heinz

7. BRUKSANVISNING.txt – Forklaring av hvilke linjer kode jeg pleier å justere ved komposisjon i Algokomp

8. IR.cva – Impulsrespons for bruk av konvolusjonsklang

9. IR2.cva – Enda en Impulsrespons



NTNU

Kunnskap for en bedre verden