

Laboration 6 – Ljuddesign för miljöer

I denna laboration ska ni använda ljudprogrammeringsmiljön SuperCollider (SC). SC är en realtidssyntesmiljö med stora möjligheter att koda ljudsyntes och signalbehandling. SC finns för Mac, Linux och Windows och kan laddas ned här:

<https://supercollider.github.io/>

Uppgiften

Uppgiften går ut på att skapa en ambient musiksonifiering till en lobby för att skapa en föränderlig musik som påverkas av tidpunkt på dagen och olika data. Er uppgift är att skapa procedurellt ljud för den information som skickas till SuperCollider via OSC.

Den första uppgiften är att bestämma vilken typ av miljö ni skapar ljudet för. Är det ett kaffe med bönor från hela världen för de som verkligen vill sitta och njuta av en kopp riktigt gott kaffe, eller är det en trendig hotellobby, eller är det ett snabbmatställe med intressant japansk-mexikansk fusion? Eller vad är er idé? Tänk på att det är er idé ni ska designa ljuden för.

Funktionen i ljudet ska vara:

1. Tydliga skillnader i ljud/musik mellan olika tidpunkt på dagen, en sonifiering av tiden alltså:
 1. Morgon (klockan 8-11)
 2. Lunch (klockan 11-13)
 3. Eftermiddag (klockan 13-17)
 4. Kväll (klockan 17-20)
 5. Happy hour (klockan 20-23)
2. En sonifiering när antalet personer ökar/förändras i lobbyn.
3. En sonifiering av temperaturen i lokalen. Temperaturen påverkas av både tidpunkt på dagen och antal personer i lokalen.

Inför laborationen

Förberedelse – 1. Ladda ner "animationen"

Ladda ner gränssnittet i rätt version för er dator. Ni kan behöva installera Java, men det kommer datorn förvarna er om. Gränssnittet är kanske inte så vackert, men ni känner igen stilen från laboration 3. Animationen går att pausa och reseta, men jag rekommenderar nog trots allt att ni stänger av och startar om den för att få SuperCollider att hänga med bra i den inkommande datan. Sorry, jag är ingen gränssnittshacker uppenbarligen. Gränssnittet skickar all data över OSC, över följande adresser "time", "person", och "temperature". Och data skickas över NetAddr(127.0.0.1, 57120).

Läs igenom denna artikel:

<https://www.soundeffects.dk/article/view/124202/171132>

Och, utforska ljuden som finns i zip-filen på kurswebben och som kan komma vara användbara. Fler ljud hittar ni exempelvis här:

<https://freesound.org/>

<https://www.looperman.com/>

Förberedelse – 2. Bekanta er med SuperCollider-koden

Grunden i SuperCollider-koden känner ni igen från tidigare laborationer.

Synthdefinitioner

Koden har sex synthdefinitioner givna. De två första, `chordSynth` och `bassSynth`, är helt procedurella och innehåller en enkel vågform, ett filter, och en envelopgenerator. Dessa behövs med stor sannolikhet anpassas och förbättras för att passa er designidé.

Därefter kommer två synthdefinitioner som spelar upp ljudsamplingar. Båda dessa har en `doneAction: 2 i PlayBuf.ar`. Det innebär att när uppspelningen av ljudsamplingen når slutet så tas instansen av synten bort från servern. Fördelen med detta är att vi kan ropa på `doorbellSamplePlayer` flera gånger i rad utan att ljuduppspelningen avbryts när ett nytt anrop kommer, och att vi inte fyller servern med tysta syntar som vi inte använder mer.

Den andra sampleuppspelaren, `bellSamplePlayer`, har ett filter vi inte diskuterat så mycket på kursen. Det är ett shelvingfilter, ett high-shelf. Ett shelvingfilter kan funka som ett "vanligt" filter (som LPF eller HPF), men kan både förstärka och dämpa frekvenser. I grundkoden ni har fått används ett `BHiShelf`, och det skrivs `BHiShelf.ar(in, freq: 1200.0, rs: 1.0, db: 0.0, mul: 1.0, add: 0.0)`. Det som är intressant att tänka på här, och som är annorlunda de "vanliga" filtren är `db`. Genom att använda ett positivt värde här förstärks frekvenserna över brytfrekvensen (för ett high-shelf) eller under brytfrekvensen (för ett low-shelf), och genom att ange ett negativt värde dämpas frekvenserna (som på ett LPF eller HPF).

Experimentera gärna med olika inställningar på filtret, eller testa `BLowShelf`. Som vanligt kom ihåg att inte sätta 0Hz som brytfrekvens och 0 som resonans (rs). Hur låter det, och varför?

Läs mer här:

<https://sv.wikipedia.org/wiki/Equalizer>

<https://www.mixinglessons.com/shelving-filter/>

<https://mynewmicrophone.com/audio-shelving-eq-what-are-low-shelf-high-shelf-filters/>

Alla dessa fyra synthdefinitioner har ett inputargument `bus`, och `Out.ar` har detta argument i stället för default som är 0. Detta görs för att från klientsidan tala om för synthdefinitionerna att skicka sitt ljud ut till synthdefinitionen `outputEffects`.

Synthdefinitionen `outputEffects` har två input argument, den första är bussen som den här synthdefinitionen tar emot ljud på, och den andra är dit den skickar sitt utgående ljud. Sedan innehåller den ett reverb och ett eko, inkommande ljud och effekterna mixas sedan ihop och skickas därefter till outputkanalen.

De nya synthdefinitioner som ni skapar kan ni sedan välja att skicka till `outputEffects` eller `outputChannel`. När jag experimenterade så skickade jag till exempel trumloopar direkt till utgången, för de blev alldeles för röriga och stökiga med rumseffekter på. `FreeVerb` är en ganska enkel efterklangsfunktion, reverb, som låter ok. Det finns andra reverb att utforska, som `GVerb` och `JPverb`. Sök i SuperCollider efter reverb så hittar ni fler. Ekot, `Greyhole`, låter

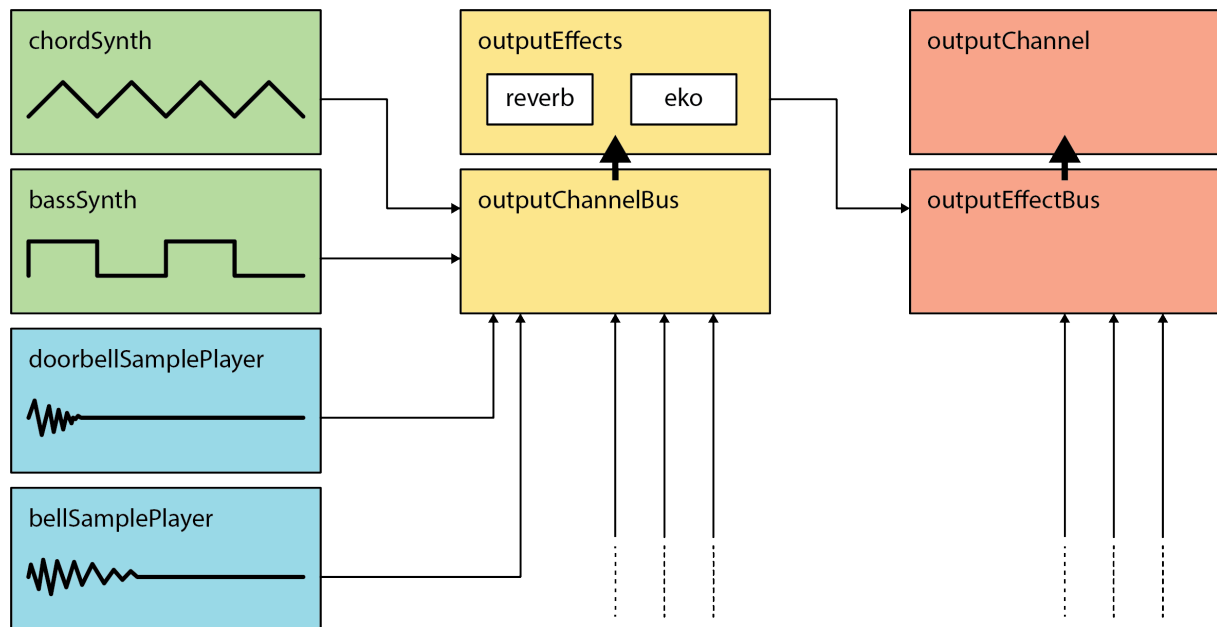
kanske inte som ett vanligt eko, men har en del intressanta egenskaper att testa på att utforska.

Läs mer här:

<https://doc.sccode.org/Classes/FreeVerb.html>

<https://doc.sccode.org/Classes/GVerb.html>

<https://doc.sccode.org/Classes/Greyhole.html>



Klientsideskript

På klientsidan skapas två bussar, `~outputChannelBus` och `~outputEffectBus`. Dessa bussar är ingångarna till `outputChannel` och `outputEffects`. När sedan instanserna av `outputEffects` och `outputChannel` skapas skickas inputbussarna med, och i `outputEffects` skickas dessutom outputbussen med. De olika outputbussarna som man vill använda behöver sedan också skickas till varje enskild synthdefinition så att ljudet får rätt väg.

Ackord och bastoner

Jag har valt fyra ackord i grundkoden och det finns tre varianter av ackordföljd: en lite mer "jazzig", en något mer poppig och glad (kanske), och en lite mer dyster (?). Både ackordljudet och basen behöver nog ha (lyssna och bestäm ni) samma variant av ackord/toner. Ni får gärna göra om och byta och ändra så att det låter bra. När ackorden har skapats har jag använt MIDInotnummer för att slippa hålla koll på de korrekta frekvenserna för tonerna, och jag har stoppat de tre tonerna i varje ackord i ett `Dictionary`. Där `key 0` innehåller en array med de tre tonerna för just det ackordet, osv.

Läs mer här:

<https://doc.sccode.org/Classes/Dictionary.html>

Därefter skapar jag ett nytt tomt `Dictionary`, `~chordSynthDictionary`, som ska innehålla tre syntinstanser, en per ton i ackordet. Poster i ett `Dictionary` läggs till med `add`, och jag använder en `for-loop` där syntinstanserna läggs till på `index` position. Därefter har jag en variabel `~chordLevel` som jag senare använder för att sätta ljudvolymen på ackordet.

När ljudet senare ska förändras för ackordet, används en `for-loop` som går igenom

```
~chordSynthDictionary och gör set på varje position. Så här:  
~chordSynthDictionary.at(index).set(\level, 1);
```

Jag har valt fyra bastoner (se kommentaren om ackord ovan) som jag tycker passar till ackorden och stoppat i en array. En instans av bassynten skapas sedan, och en ljudvolymvariabel deklarerar och sätts till 0, för i början vill jag inte att basen ska höras.

Ljudsamplingar

I laboration 1 lästes ljudsamplingar in med `Buffer.read`, och det är samma approach i den här laborationen. Jag använder två samplingar för att markera när personer kommer in och när de olika tidpunkterna ska markeras, men det finns många fler sätt att använda ljudsamplingar på. Se till att sökvägen stämmer i grundkoden och där ni har lagt ljudfilerna.

De samplingar som jag lag till i zip-filen är anpassade efter den mer jazziga ackordföljden. De kanske inte passar alls bra till de andra ackordföljderna. Så, om ni väljer att jobba med de andra kan det vara bra att ladda ner nya samplingar från nätet (se nedan i detta dokument). Den "poppiga" ackordföljden är C, G, A, F, och den mer "dystra" ackordföljden är "G, B, C, Cm".

Klocka och sequencer

Sedan följer ett antal rader kod som fungerar som en sequencer och som spelar upp ackorden och bastonerna. Tempot styrs av `TempoClock` som använder variabeln `~tempoInBPM` som är satt till 75, delat med 60. Tyvärr stannar inte sequencern när `GIUt/animationen` pausas. Det hade varit smart, men min tid har inte riktigt räckt till för att göra en perfekt app för att skicka data.

Läs mer här:

<https://doc.sccode.org/Classes/AppClock.html>

<https://doc.sccode.org/Classes/TempoClock.html>

I funktionen, `~sequencingFunction`, loopas ackordsyntarna igenom för att varje syntinstans ska ha rätt ton för ackordet. I stället för en for-loop använder jag i det här fallet `.size` för att få ut storleken på syntdictionaryt och sedan `.do` för att loopa rätt antal gånger. Resterande del av funktionen håller koll på vilket taktslag som vi är på, räknar upp till 8 taktslag per ackord (två takter per ackord alltså), kollar vilket ackordnummer som spelas, och om animationen nått sitt slut.

Tidslyssnare

Därefter kommer en OSC-lyssnare, som lyssnar på adress `time`. Den lyssnaren startar först klockan för sequencern, och därefter håller koll på timmen som kommer som meddelande, kollar om det är en ny timme eller inte, och om det är det anropas funktionen

`~hourSyncedStuff`.

I den efterföljande funktionen används en `switch` för att det är enklare och smidigare än if-satser i SuperCollider. Läs mer här:

<https://doc.sccode.org/Reference/Control-Structures.html>

Här förändras ljudvolymen för bassynten och ackordsyntarna för de olika tidpunkterna, och funktionen, `~playbell`, för att spela upp klockljudet anropas. När klockan är 23 stänger stället och `~stopFlag` sätts till `true`.

Det är i den här funktionen som ni ska skriva/anropa allt det som ska påverka musiken/ljuden i relation till tidpunkterna.

Personlyssnare

Sedan kommer en till OSC-lyssnare som lyssnar på adress `person`, och som spelar upp ett ringklockeljud varje gång en person kommer in i lobbyn med hjälp av variabeln

`~numberOfPersons`.

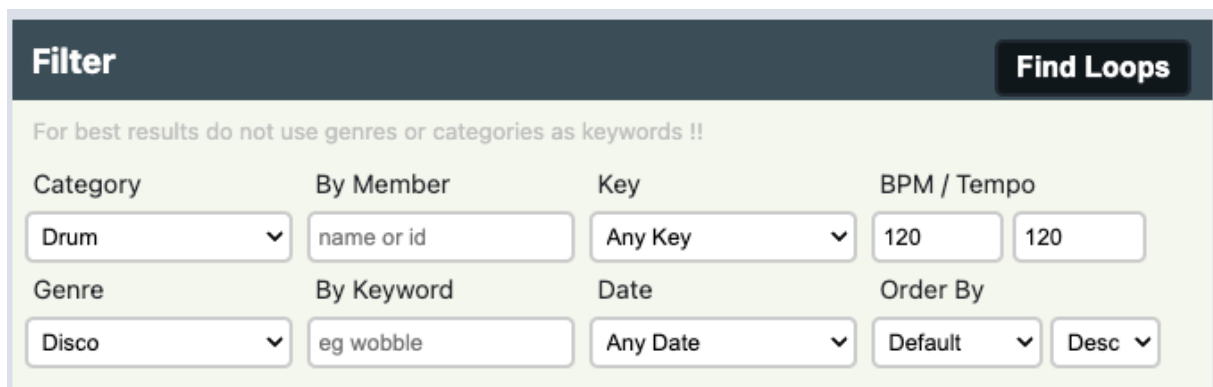
Temperaturlyssnare

Och till slut finns en OSC-lyssnare för `temperature`, och som förändrar ljudvolymen på `outputChannel`. Det sker med en linjär till linjär mappning för input-temperaturer mellan 19.7 och 22.3 mappat till 0.3 och 1.

Att använda nya samplingar

I zip-filen finns ett antal samplingar ni kan använda. Där finns ljudeffekter som kan användas som morgonljud eller kvällsljud, och det finns olika instrument och trumloopar. Dessa matchar ganska bra tonarten som jag har valt (dvs ackorden i `~chordDictionary`) och tempot på 75 slag per minut (dvs `~tempoInBPM`). Men, välj gärna andra tonarter och ackord, testa att arbeta med andra tempon. Skapa en ljudmiljö som matchar er designidé.

När det gäller tempo kan man ha ett långsamt tempo på morgonen (kanske 60bpm) för att sedan under happy hour använda en trumloop som går dubbelt så snabbt i 120bpm. Då kommer allt ljud att snyggt synka mot varandra. När ni tittar på samplingar på Looperman behöver ni registrera ett konto, men annars är det gratis att använda. Här kan ni sedan välja kategori (Category) för att till exempel söka bland trumloopar, med Key väljer ni tonart (mina ackord börjar med Cm, dvs C moll), BPM/Tempo sätter tempot, med Genre väljer ni musikstil, osv.



The screenshot shows the 'Filter' section of the Looperman website. It features a dark header with the word 'Filter' on the left and a 'Find Loops' button on the right. Below the header, a light green box contains the text 'For best results do not use genres or categories as keywords !!'. The filter options are organized into two rows. The first row includes 'Category' (set to 'Drum'), 'By Member' (set to 'name or id'), 'Key' (set to 'Any Key'), and 'BPM / Tempo' (set to '120'). The second row includes 'Genre' (set to 'Disco'), 'By Keyword' (set to 'eg wobble'), 'Date' (set to 'Any Date'), and 'Order By' (set to 'Default'). Each option is presented in a dropdown menu or a text input field.

I freesound kan ni söka efter ljudeffekter, som sorl från människor, eller ljud från en dörr som öppnas när det dyker upp personer i animationen. Ibland har dessa ljud onödig tystnad i början eller slutet vilket ni kan vilja att redigera bort. Ljudfiler går bra att redigera i Audacity som är gratis att ladda ner.

<https://www.audacityteam.org/>

Det tempo jag har valt, 75bpm, matchar inte animationen perfekt. Det gör inget tycker jag. Tänk er att det vi skapar är en kontinuerligt föränderlig ljudmiljö som skapar en bakgrundsmusik som spelas hela dagen. Ingen kommer att lyssna efter eller höra om de olika förändringarna sker perfekt i synk med ett tempo. Tänk i stället att förändringarna ska komma när datan förändras.

Ändra tonart och ackord

När ni designar för ert "ställe", vill ni säkert också ändra på ackorden och tonerna. Ni kanske också vill göra det lite mer intressant än enkla tretonsackord och ändra till fyra eller fem toner. Helt ok. Experimentera och ha kul!

<https://www.schoolofrock.com/resources/keyboard/piano-chords-for-beginners>

Kom ihåg de andra laborationerna

Kom ihåg det som gått igenom och det ni har gjort på de andra laborationerna. Kom ihåg att:

- skicka gate-signaler för att trigga envelopgeneratorer
- ändra klangfärg och vågform, `SinOsc`, `LFTri`, `LFPulse`, `LFSaw` osv
- utnyttja filter (LPF, BPF, HPF, BRF, och shelvingfilter) för att färga ljudet och förändra innehållet
- modulera signaler med andra signaler för att skapa förändringar och effekter
- använda envelopgeneratorer (som `perc` eller `adsr`) för att skapa amplitudförändringar och mer intressanta ljud
- brus kan vara användbart som byggsten i många olika typer av ljud
- SuperCollider har många intressanta möjligheter som `Klank` eller `Crackle` eller `LFGauss`
- använda `.lag` för att göra snabba förändringar långsammare
- förändra ljudvolymen för att skapa mer dynamik och koppla mot data
- använda `.range` för att justera ljudparametrar
- mappa data och ljudparametrar exempelvis genom `linlin` eller `linexp`
- använda rätt rate, dvs `.ar` för audiorate och `.kr` för controlrate
- `doneAction 2` tar bort en syntinstans från servern (på gott och ont)
- lyssna, och låter det bra så är det rätt, låter det illa så gör om (om inte illa var själva poängen...).