Laboration 1 – Foley

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/

Laborationen går ut på att spela in ett ljud och sedan använda SC för att procedurellt ändra ljudet relativt en (fiktiv) interaktion.

SC är en kodmiljö som består av två sidor, en serversida som innehåller synthdefinitioner som skapar ljud och en klientsida som exempelvis skickar noter till synthdefinitionerna för att förändra tonhöjden på syntarna.

Uppgiften

Uppgiften går ut på att göra fyra ljud till en kaffemaskin. Kaffemaskinen har endast en knapp och ingen visuell återkoppling, därför behövs ljud. Knappen togglar kaffestyrka (svag, medel, stark) vid korta tryck, medan ett långt tryck startar bryggningen. Ljudet ska användas för att sonifiera och återkoppla kaffestyrkan och att bryggning startas. Alltså behövs fyra typer av ljudåterkoppling till användaren.

Inför laborationen

Förberedelse – 1. Spela in några olika ljud

Till uppgiften behöver ni några olika ljud att arbeta med. Fundera över hur de fyra ljuden ska låta, eller förändras för att återkoppla till användaren. Ska det vara korta ljud eller längre, ska de vara mer sprakande brusiga ljud eller klingande toner, vilket frekvensomfång ska de ha, och vilket dynamiskt omfång?

Klura sedan vad ni har omkring er som kan användas för att skapa grunden till ljuden ni vill använda; knäpp med fingrarna, bryt av en liten gren, slå två trägafflar mot varandra, klinga en sked mot ett glas, riv sönder ett papper, dra lite fuktig frigolit mot en glasruta, …

Spela sen in några ljud. För ändamålet räcker det gott med att spela in i mobiltelefonen. Kolla bara upp att telefonens inspelning har tillräckligt kvalitet vad gäller samplingsfrekvens, bitupplösning och kodning. Kom ihåg Nyquist-Shannon-teoremet och undvik förstörande komprimering som mp3 och m4a.

Förberedelse – 2. Bekanta er med SC

Ladda först ned och installera SC: https://supercollider.github.io/download

När SC startas skapas ett nytt tomt ”projekt”. Till höger (oftast) om detta ligger Help browser där det går att söka hjälp om SC, och nedanför detta finns Post window. Nedanför Post window visas om kodtolken är aktiv, och en del info om Servern som CPU-belastning etc.

Det är möjligt att skriva ut saker till Post window, och det görs med `postln` (post line). Genom `postln("Hello world");` så skriver SC ut Hello world i Post window. Kodraden exekveras genom att stå på kodraden med markören och klicka på <Shift> + <Enter>. Ett annat sätt att skriva detta på är `("Hello world").postln;` vilket resulterar i samma output.

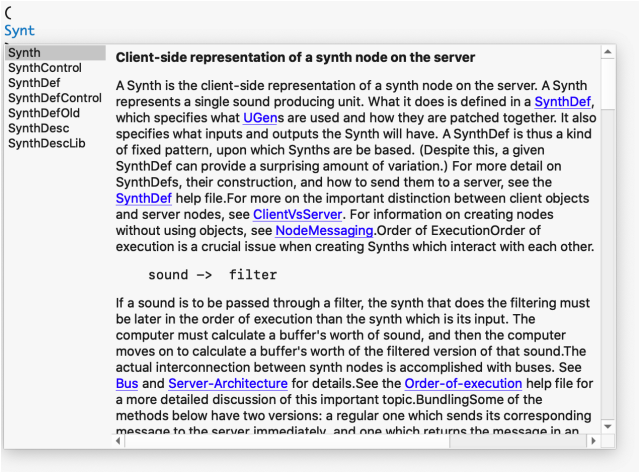
En variabel deklareras med `var x;` och flera variabler kan deklareras i samtidigt med `var x, y, z;`. Alla variabler måste deklareras först i en funktion eller i en synthdefinition. Globala variabler deklareras utanför funktionerna och det finns ”superglobala” variabler typ s (server) och w (window), och det finns environment-variabler som deklareras med `~`.

Skapa en variabel som innehåller strängen ”Hello world” och skriv sedan ut variabeln. Eftersom <Shift> + <Enter> exekverar en rad i taget så måste båda raderna markeras för att innehållet i variabeln ska skrivas ut. Därför är det bra att sätta parenteser kring koden, en startparentes innan koden och sedan en slutparentes efter koden. Då går det att använda <Ctrl> + <Enter> i Windows eller <Cmd> + <Enter> i Mac för att exekvera all kod i parentesen. Med <Shift> + <Enter> körs fortfarande bara den/de rader som är markerade. Med <Ctrl> + <.> i Windows och <Cmd> + <.> stoppas exekveringen.

```
(
var x = ("Hello world");
x.postln;
)
```

Med `//` kommenteras en rad i taget i koden, och med `/* .. */` kommenteras flera rader.

När man skriver kod och vissa keywords dyker det upp rutor så att det är enkelt att välja rätt nyckelord och få tips om användning och fortsättning på koden:



Det händer att kodtolken i SC rasar, under menyn Language kan man både avsluta den och/eller starta om den.

Att skriva ut till Post window görs på klientsidan. Det går inte att skriva ut på samma sätt från serversidan. Än så länge har inte servern startats, detta måste göras manuellt antingen genom att välja Server -> Boot Server eller genom att exekvera `s.boot;`. I menyn Server kan man också starta om servern eller döda alla servrar som är i gång, det går också att stänga av servern genom att exekvera `s.quit;`. När servern bootats visas i Post window de ljuddevices som är tillgängliga i datorn. När kodtolken startas om dödas också alla servrar som är igång.

Med `s.scope;` skapas ett oscilloskop vilket kan vara till hjälp i laborationerna i den här kursen.

Läs mer här:
https://doc.sccode.org/Classes/Server.html

Testa på SC och bekanta er med gränssnittet i SC. Följande är några tips för att komma igång mer än vad den korta introduktionen ger i denna text.

Eli Fieeldsteel har en väldigt bra Youtube-kanal med många exempel och tips:
https://youtu.be/yRzsOOiJ_p4

Det finns också ett antal tutorials här:
http://doc.sccode.org/Tutorials/Getting-Started/00-Getting-Started-With-SC.html

Och här finns info om SC, om strukturen och klasser:
http://doc.sccode.org/

Spela upp ljudfiler

Ett inspelat ljud spelas upp med hjälp av en `Buffer` till vilken man läser in ljudfilen. Detta görs på klientsidan med `Buffer.read(server, path, startFrame: 0, numFrames: -1, action, bufnum)`. Där `server` normalt är `s` och `path` är sökvägen till ljudfilen. `Buffer` funkar bara med wav-filer.

Buffer ska vara fylld med en ljudfil för att uppspelning ska funka. Så här:

```
var soundSample = Buffer.read(s,
thisProcess.nowExecutingPath.dirname++"music.wav");
```

Därefter skickas bufferten till synthdefinitionen med:

```
Synth.new(\samplePlayer, [\bufnum, soundSample]);
```

Läs mer här:
https://doc.sccode.org/Classes/Buffer.html

På serversidan måste en synthdefinition finnas som tar emot ljudsamplingen. Samplingen tas emot med ett inputargument `bufnum`. I synthdefinitionen behövs kod, en sampleplaybackoscillator, för att spela upp ljudet från buffern. Det görs med `PlayBuf`.

Läs mer här:
https://doc.sccode.org/Classes/PlayBuf.html

Sampleplaybackoscillatorn stoppas i `sound`, och `sound` i sin tur stoppas i `output` och `output` skickas sedan till ljudutgången med `Out.ar`.

Ändra tonhöjd på ett inspelat ljud

Det är enkelt att ändra tonhöjd eller uppspelningsfrekvens av ljudet. Det görs genom att lägga till en faktor i synthdefinitionen på raten där `rate` ställs in. Om faktorn som ni använder för att multiplicera är 2, så spelas ljudet upp dubbelt så snabbt vilket innebär en oktav högre.

Experimentera med olika faktorer, till exempel 0.5, 2, 10, 0.1. Hur låter det, och varför?

_____

_____

_____

Ett sätt att ändra detta smidigare är genom att deklarera en ny input i synthdefinitionen, exempelvis `rateMod = 1`, som sedan används på raden med `rate:`. På klientsidan kan sedan raden med `Synth.new` stoppas in i en environmentvariabel, till exempel `~samplePlayer`, där `~` är det som deklarerar den som environment. När detta är gjort går det att komma åt `rateMod` med en rad klientsidekod:

```
~samplePlayer.set(\rateMod, 0.2);
```

Nu kan det vara läge att ändra `loop: 0,` till `loop: 1`, så kan ni tydligare höra hur frekvensen i ljudet ändras.

Att ändra ljudvolym på ljudet

Det enklaste sättet att ändra ljudvolymen på ett ljud i en synthdefinition är att multiplicera det med ett värde. Exempelvis:

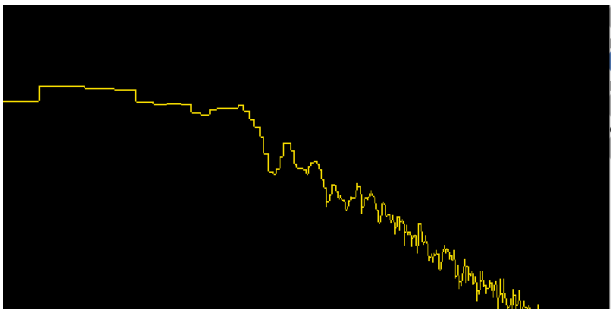
```
var output = sound * 0.5;
```

Då halveras ljudvolymen. I stället för att skriva ett explicit värde kan vi använda en variabel till vilken vi tar emot ett inputargument och så kan vi förändra ljudvolymen på samma sätt som uppspelningsfrekvensen ändrades ovan.

Något som kan vara bra är att använda `.lag(x)` för att lägga en fördröjning på en ändring. Hur stor den fördröjningen (`x`) ska vara får man experimentera sig fram till. Men, om ni har en förändring av brytfrekvens för ett filter som är väldigt stor och snabb, brytfrekvensen hoppar från 4000Hz till 100Hz, så uppstår lätt missljud eller knäpp. Då kan en fördröjning av förändringen underlätta.

Lågpass- och högpassfiltrering av ljudet

Ett bra sätt att ändra hur ett ljud låter eller för att ge ett och samma ljud olika egenskaper för olika funktion i ett gränssnitt är att filtera ljudet. Testa först ett lågpassfilter (LPF). Ett lågpassfilter dämpar frekvenser högre än brytfrekvensen i filtret. Ett lågpassfilter skapas med `LPF` och normalt ligger det i audio rate (`ar`) och skrivs: `LPF.ar(in: 0.0, freq: 440.0, mul: 1.0, add: 0.0).LPF` är ett andra ordningens Butterworthfilter.



Läs mer här:
https://doc.sccode.org/Classes/LPF.html
https://en.wikipedia.org/wiki/Low-pass_filter
https://en.wikipedia.org/wiki/Butterworth_filter

Argumentet `in` är det summerade ljudet, `freq` är brytfrekvensen för filtret. OBS! Ange aldrig 0 i brytfrekvens. De övriga argumenten känner ni igen, och behövs normalt inte anges. Testa med en brytfrekvens på 440Hz.

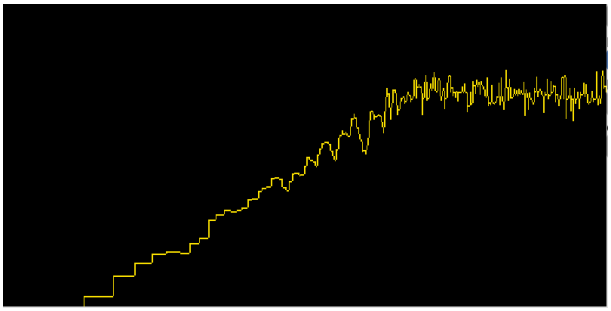
Hur låter det? Testa olika brytfrekvenser, hur påverkar det ljudet?

_____

_____

_____

Testa sedan att ändra LPF till ett högpassfilter (HPF). Ett högpassfilter dämpar frekvenser lägre än brytfrekvensen för filtret.



Hur låter det? Testa andra brytfrekvenser, hur förändras ljudet?

_____

_____

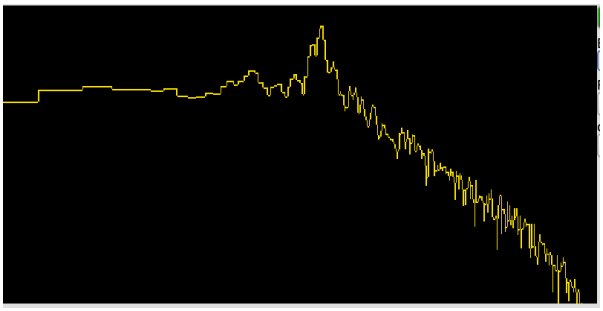
_____

Läs mer här:
https://doc.sccode.org/Classes/HPF.html
https://en.wikipedia.org/wiki/High-pass_filter

Resonans i filter

Det går också att använda resonanta filter i SC. Det innebär att en del av utsignalen från ett filter återmatas till ingången. Detta kan skapa en självsvängning i filtret, där filtret genererar en sinusvåg i brytfrekvensens frekvens. I SC kan man göra detta med `RLPF` respektive `RHPF`. Dessa skrivs som `RLPF.ar(in: 0.0, freq: 440.0, rq: 1.0, mul: 1.0, add: 0.0)` där argumentet `rq` står resonansen eller med andra ord relationen mellan bandbredd och brytfrekvens, och anges mellan 0 och 1 där 0 är max resonans och 1 är ingen resonans. OBS! Ange aldrig 0 i `rq`.

Läs mer här:
https://doc.sccode.org/Classes/RLPF.html
https://doc.sccode.org/Classes/RHPF.html



Testa att använda ett resonant filter och förändra värdet på `rq`. Hur låter det?

_____

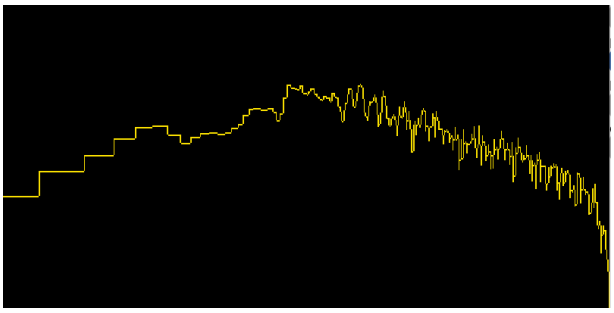
_____

_____

Läs mer här:
https://en.wikipedia.org/wiki/Resonance

Bandpassfilter

Ett annat filter som kan vara användbart är bandpassfilter, `BPF`. Ett bandpassfilter släpper igenom ett band av frekvenser över och under brytfrekvensen. Det skrivs så här: `BPF.ar(in: 0.0, freq: 440.0, rq: 1.0, mul: 1.0, add: 0.0).BPF` är också ett andra ordningens Butterworthfilter.



Läs mer här:
https://doc.sccode.org/Classes/BPF.html
https://en.wikipedia.org/wiki/Band-pass_filter

Testa att använda ett bandpassfilter och förändra brytfrekvenserna i `range`. Experimentera också med andra värden på `rq`. Hur låter det, och hur ser frekvensanalysen ut?

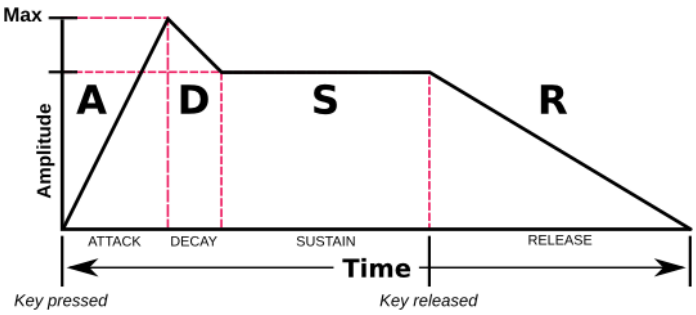
_____

_____

_____

Skapa en envelop

En envelop är ljudets volymkontur, och det klassiska exemplet är ADSR-envelopgeneratorn som består av Attack, Decay, Sustain, Release. Attack är tiden det tar från att en tangent trycks ned på en synt tills ljudvolymen når max, Decay är tiden det tar från att attackfasen är klar till ljudet når sitt viloläge dvs Sustainnivån, och Release är tiden det tar efter att tangenten släpps upp tills ljudet är helt tyst.



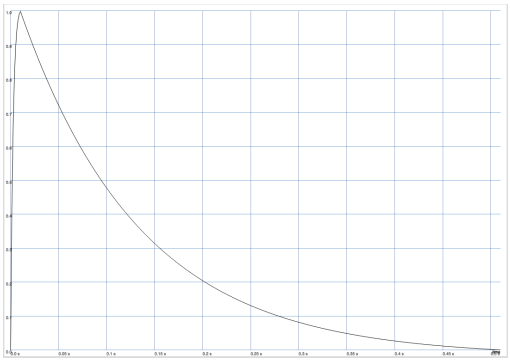
I SC finns det många olika envelopformer att välja mellan. För den här uppgiften kan nog `perc` passa bäst. Den är enklare än ADSR och har bara Attack och Release. Man skapar en envelop med `EnvGen`. Men, det skulle gå att skapa en ganska komplex vågform om envelopen loopades och justerades till att svänga mellan -1 och 1. En envelop är normalt en kontrollsignal, och därför är `EnvGen` oftast i control rate `.kr`. EnvGen skrivs så här: `EnvGen.kr(envelope, gate: 1.0, levelScale: 1.0, levelBias: 0.0, timeScale: 1.0, doneAction: 0)`, och det viktigaste här är inputargumentet för `envelope` och `gate`.,

där `envelope` är själva formen på envelopen (antingen en `Env`-instans eller en kontrollarray) och `gate` är kontrollsignalen om tangenten på klaviaturet är nedtryckt (1) eller inte (0).

Läs mer här:
https://doc.sccode.org/Classes/EnvGen.html

Med en instans av `Env` skapas själva konturen av envelopen. Denna skrivs som `Env.new(levels: [ 0, 1, 0 ], times: [ 1, 1 ], curve: 'lin', releaseNode, loopNode, offset: 0)`, men det finns ett antal versioner på `Env` som underlättar arbetet att skapa nivåer och tider, exempelvis ADSR.

En perc skrivs: `Env.perc(attackTime: 0.01, releaseTime: 0.5, level: 1.0, curve: -4.0)`. Låt oss bryta isär detta exempel. Attacktiden är 0.01 sekunder, det är en kort attacktid. Att ange 0 som attacktid kan resultera i korta klickljud när ljudet startar. Det kommer alltså ta 0.01 sekunder för ljudet att gå från tyst till max ljud. När tangenten på klaviaturet släpps upp och ljudet ska slutar tar det 0.5 sekund innan ljudet är helt tyst. Ljudnivån, `level`, är 1, dvs nivån som är max för ljudet. Envelopens krökning styrs av `curve`, där 0 är en linjär kurva, medan värden över eller under 0 skapar logaritmisk respektive exponentiell krökning. Genom att skriva `.plot` efter en `Env` så öppnas ett fönster där envelopens kontor ritas ut. Här nedan plottas ADSR-exemplet ovan, men med en Sustain-tid som är 0 då ingen tangent blir nedtryckt eller simulerad vid plot.



Läs mer här:
https://doc.sccode.org/Classes/Env.html

Skriv i exemplet på ADSRen ovan i synthdefinitionen från förra laborationen. Tänk på att argumenten inte behöver specificeras om de skrivs i rätt ordning, och argument som inte ges blir default:

```
var envelope = EnvGen.kr(Env.perc(attackTime: 0.01, releaseTime: 0.5,
level: 1.0, curve: -4.0));
```

Jag föredrar att lägga envelopen i en variabel i stället för att skriva den direkt på oscillatorn. Envelopen lägger ni sedan på ljudet i synthdefinitionen genom att multiplicera ljudet med envelopen.

```
output = output * envelope;
```

Hur låter det? Hörs det att envelopen påverkar ljudets volymmässiga kontur?

_____

_____

_____

En envelop kan också användas för att förändra brytfrekvensen på ett filter. Normalt går en envelop från 0 till max (1) och till 0 igen, och vi vill inte använda en brytfrekvens på 0Hz. Så ett sätt att påverka brytfrekvensen i ett filter är att använda `.range(x, y)`. Exempelvis så här:

```
var envelope2 = EnvGen.kr(Env.perc(attackTime: 0.01, releaseTime: 0.1,
level: 1.0, curve: -4.0).range(100,10000));
```

Om envelope2 sedan används i stället för en specifik brytfrekvens, kommer filtret svepas från 10000Hz och ned till 100Hz på 0.1 sekunder.

Hur låter det? Hörs det att envelopen påverkar ljudets frekvensmässiga innehåll över tid?

_____

_____

_____

Samma princip går förstås också att använda för att förändra uppspelningshastigheten.

Spela upp förändringar av ljudet

Som ett sista steg kan det vara relevant att spela upp sekvenser av förändringar över tid, exempelvis om man vill förändra tonhöjden för de olika funktionerna och automatisera detta.

Ett sätt att lösa detta på är genom en `fork`. En `fork` är en förgrening av processerna som snurrar, och bildar en egen process som körs ”parallellt” med övriga processer. En fork skrivs:

```
fork({
    kod som ska köras;
});
```

I `fork`en kan man sedan använda `.set()` för att skicka argument till synthdefinitionen på servern. Dessa argument kommer att skickas så snabbt att man inte kommer höra hur de förändrar ljudet, därför kan man använda `.wait` för att få processen att vänta mellan att argumenten skickas. Så, `1.wait;` väntar processen en sekund medan `0.1.wait;` väntar processen en tiondels sekund, exempelvis:

```
fork({
    Synth.new(\samplePlayer, [\bufnum, soundSample]);
    0.15.wait;
    Synth.new(\samplePlayer, [\bufnum, soundSample, \rateMod, 0.75]);
});
```

Spara ljudet från SC

Ljud som skapas och förändras i SC kan sparas ned som ljudfiler med hjälp av `Server.default.record`. Inspelningen stoppas sedan med `Server.default.stopRecording`. Eftersom vi oftast använder `s` för servern räcker det med att man skriver `s.record;`. Det går också att skicka med olika inputargument: `.record(path, bus, numChannels, node, duration).`

Läs mer här:
https://doc.sccode.org/Classes/Recorder.html

Ni ska spela upp de olika ljuden vid seminariet efter laborationen.