Laboration 2 – Procedurellt ljud

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 göra ett antal procedurella ljud i SC. Dels imitation av gammal trummaskin från början av 80-talet, dels en form av rymdeffekt från ett gammalt datorspel, och sedan två versioner av procedurellt skapade ljudmattor/musik.

Inför laborationen

Förberedelse – 1. Fortsätt att bekanta er med SC

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/

Förberedelse – 2. Kolla upp några gamla trummaskiner

Kolla upp information om några olika gamla trummaskiner och fundera på hur deras ljud kan framställas procedurellt. Exempelvis:

Boss DR-110
https://www.vintagesynth.com/roland/dr110.php
Roland TR-606
https://www.vintagesynth.com/roland/606.php

Korg Minipops
https://www.vintagesynth.com/korg/minipops120

Skapa trumljud - bastrumma

Det är fyra trumljud som ska skapas: bastrumma, virveltrumma, stängd HiHat och öppen HiHat. Börja med bastrumman, grunden finns redan i exempelkoden. En bastrumma består av två ljud, dels själva kroppen (basljudet), dels ett klickljud som uppstår när pedalen träffar trumskinnet. I exempelkoden finns tre variabler definierade, `body`, `hit` och `output`. Alla variabler i SC måste deklareras först i en funktion eller synthdefinition. Själv trumljudet är en sinuston, i mitt exempel 75Hz. Tonen har kort attack och en ganska kort release, 0.1 sekunder. Slutet av synthdefinitionen avslutas med `.play;` då spelas ljudet upp när synthdefinitionen läses över till servern, normalt används `.add;`.

Hur låter det? Låter det som en bastrumma? Experimentera med frekvensen och releasetiden.

_____

_____

För att få lite mer närvaro i attack i ljudet bör klickljudet skapas. Jag gjorde det genom att använda en triangeloscillator (`LFTri`) som har övertoner (harmonik eller en rikare/vassare klangfärg) jämfört mot sinustonen. Frekvensen ska vara högre upp jämfört mot bastrummans kropp, exempelvis 220Hz, och releasetiden för klicket kortare än för kroppljudet, exempelvis 0.01 sekunder. För att en oscillator ska funka i SC, måste man ange om den ska vara i hörbara frekvenser, dvs `.ar` (audio rate), eller om SC kan spara beräkningskraft genom att sänka samplingsfrekvensen och använda `.kr` (control rate) för vågformer som ligger under hörbara frekvenser. Exempelvis:

```
hit = LFTri.ar(220);
```

Mixa sedan ihop de två ljuden genom att addera dem (`+`) och sänk ljudvolymen på hit-ljudet med ungefär en tredjedel.

Hur låter det? Låter det mer som en bastrumma? Experimentera med frekvensen och ljudvolymen på hit-ljudet.

I stället för triangelvåg kan man också experimentera med `LFPulse` för fyrkantsvåg, `LFSaw` för sågtandsvåg, eller kanske med ett brus (exempelvis rosa brus med `PinkNoise.ar`).

Virveltrumma

Skriv en ny synthdefinition för virveltrumman. En virveltrumma består också av två ljud, dels själva kroppen, dels sejarmattan på undersidan som skapar det smattrande ljudet. Kroppen skapar man bra genom att summera två sinusoscillatorer den ena med en halvhög frekvens, exempelvis 220Hz, och den andra 1.5 gånger högre upp i frekvens. Den ljusare oscillator bör sänkas i ljudvolym till ungefär 10% när de två sinusljuden summeras. En liknande envelop som används till bastrumman kan användas även här.

Hur låter det? Låter det som en virveltrumma? Experimentera med frekvensen och ljudvolymen på den andra oscillatorn.

Det som verkligen saknas i ljudet är sejarmattan. Det skapas genom att använda vitt brus, och sedan högpassfiltrera detta brus för att skära bort de lägre frekvenserna i ljudet. Vitt brus skapas med `WhiteNoise.ar()`, och inom parentesen kan man sätta ljudnivån, exempelvis till 0.25. Ett högpassfilter skapas med `HPF.ar(in, frekvens)`. Exempelvis så här:

```
noise = WhiteNoise.ar(0.25);
noise = HPF.ar(noise, 750);
```

Använd sedan en liknande envelop för bruset och mixa ihop (summera) de två olika ljuden.

Låter det mer som en virveltrumma nu? Experimentera med frekvensen på högpassfiltret och releasetiden på enveloperna.

Öppen och stängd HiHat

Skriv en ny synthdefinition för HiHat, enklas görs en för stängd och en för öppen HiHat. HiHat kan skapas med brus, och många gamla trummaskiner och datorer och spelkonsoller använde bara brus, men i det här exemplet ska vi både använda brus samt skapa ett metalliskt klingande ljud. Använd WhiteNoise.ar som för virveltrumman, använda också ett högpassfilter (HPF.ar) med en brytfrekvens på 1500Hz. Skapa en envelop med kort attack och kort release, exempelvis:

```
output = noise * EnvGen.ar(Env.perc(0.005, 0.025, 1, -1));
```

Låter det som en HiHat? Experimentera med frekvensen på högpassfiltret och attack- och releasetiden på envelopen.

Det som skiljer stängd och öppen HiHat åt är attack- och releasetiden. Så, det räcker bra med att skriva koden för stängd HiHat och sedan kopiera den och döpa om synthdefinitionen till openHiHat och anpassa inställningarna där.

Nästa steg i att skapa ett bättre HiHat-ljud är att skapa det metalliskt klingande ljudet. Detta görs med fyra fyrkantsoscillatorer (`LFPulse.ar`). Frekvenserna för dessa är 2000, 1150, 820 och 465Hz. Sätt pulsbredden till `0.5` (dvs 50%), summera alla fyra ljuden (till en variabel, exempelvis `tones`) och dividera med 4 för att undvika att ljudvolymen överstyr (dvs distorsion uppstår pga att summan av ljuden blir för stark).

Lyssna bara på `tones`, hur låter det?

Summera tones med bruset innan envelopen, exempelvis så här:

```
output = noise + (tones / 2);
```

Låter det som en HiHat nu? Experimentera med frekvensen på högpassfiltret och attack- och releasetiden på envelopen, och testa gärna andra frekvenser för `tones`.

Lyssna på trummorna

Nu, när det finns fyra olika synthdefinitioner, bastrumma, virveltrumma, stängd och öppen HiHat, är det dags att öppna SC-filen `drum_machine.scd`. Kolla att ni har läst över den

senaste versionen av varje synthdefinition till servern, och uppdatera koden i drum_machine så att namnen på synthdefinitionerna stämmer med era. Kör sedan koden i drum_machine.

Hur låter det? Har ni lyckats göra procedurella trumljud? Låter det som riktiga trummor, eller som en gammal trummaskin?

_____

_____

_____

Det finns risk för att summan av alla trumljud blir för stark så att det blir distorsion. Så, ni kan behöva sänka ljudvolymen på de olika trummorna. Enklast görs det genom att multiplicera ljudet ut från varje synthdefinition med 0.5 eller vad som blir lämpligt. Ni kanske också vill passa på att mixa ljuden, så att HiHaten är lite svagare än bastrumman. Med `ServerMeter` kan ni se utgångsvolymen och den ska helst inte bli röd:

```
ServerMeter.new(s, 0, 2);
```

Spara ljudet från SC

Ni ska spela upp trumljuden vid seminariet efter laborationen. Så, när ni har lyssnat på alla ljud samtidigt, gå tillbaka och förbättra dem. Kanske behövs hit-ljudet i bastrumman lyftas fram mer, kanske behöver brusets envelop kortare release så att den blir mer snärtig, och kanske behöver virveltrummans brusfilter få annan brytfrekvens, och hur är mixen och frekvenserna i tonerna i HiHaten? Skruva på ljuden och se över den totala mixen. Spara sen ned en fil med hjälp av `Server.default.record` några takter trummor. Om ni vill får ni gärna se över sequencern som spelar rytmen och ändra i den. Inspelningen stoppas sedan med `Server.default.stopRecording`.

Procedurella ljudeffekter

Nästa del i laborationen går ut på att skapa en ljudeffekt, typ ett rymdskepp som lyfter, som låter som gamla datorspel. Ofta på 80-talet hade datorerna och spelkonsolerna enkla små synthchip i sig, som SID6581 i Commodore 64, AY-3-8910 i Atari ST eller RP2A07 i Nintendo NES. Ett sätt att skapa tämligen komplexa ljud är genom att modulera ljudet på olika sätt. Det ska vi göra i den här delen av laborationen. Skapa först en ny synthdefinition, välj att den ska ha `.play` i slutet så att ljudet spelas upp direkt på servern och inte bara läses över dit.

Vi ska skapa en svepande uppåt- eller nedåtgående förändring i tonhöjd. Det gör vi med en sågtandsoscillator, och då detta ska ske långsamt använder vi `control rate` (`LFSaw.kr`). Frekvensen ska vara låg (0.1Hz) och fasen på vågformen (`iphase:`) ska sättas till 1, då börjar ljudet från lägsta punkten i vågformen. Normalt svänger ett ljud runt 0 mellan -1 och 1. I det här fallet vill vi att omfånget ska vara mellan 1 och 8.

```
var sawtooth = LFSaw.kr(0.1, iphase: 1).range(1, 8);
```

Ljudet som ska skapas är en fyrkantsvåg i `audio rate` (`LFPulse.ar`) där grundfrekvensen är 220Hz men som moduleras av sågtandsvågen. Pulsbredden sätts till 50% (ren fyrkantsvåg alltså) och ljudvolymen kan sänkas till 25% för att inte bli för stark.

```
var sound = LFPulse.ar(220 * sawtooth, width: 0.5, mul: 0.25);
```

Lyssna på ljudet. Det bör vara sakta svepande uppåt i frekvens. Prova att ändra grundfrekvens i fyrkantsoscillatorn. Prova att ändra `range` på sågtanden. Testa att ändra pulsbredden (`width`) på fyrkantsvågen.

_____

Nu ska amplituden på ljudet moduleras. Skapa därför en ny fyrkantsoscillator i `control rate`. Sätt frekvensen till 4Hz och modulera denna med sågtandsvågen och se till att omfången av fyrkantsvågen är mellan 0 och 1. Det innebär att frekvensen för fyrkantsoscillatorn ska gå från 4Hz och sedan öka till 32Hz. Låt den här fyrkantsvågen (`soundLevelMod` i min kod) sedan amplitudmodulera ljudet från den första fyrkantsoscillatorn (`sound` i min kod) så här:

```
var output = sound * soundLevelMod;
```

Många av de gamla synthchipen hade någon form av brusgeneratorer. Skapa därför ett vitt brus och sätt dess volym till 20%

```
var noise = WhiteNoise.ar(0.2);
```

Brusets ljudvolym ska sedan förändras av en envelop. Använd samma typ av envelop som för trumljuden, med kort attacktid (0.01), relativt lång release (2) och sågtanden ska användas som trigger som ser till att envelopen startar om varje varv. För att göra det används `gate` och vi anger sågtanden minus 1.

```
var envelope = EnvGen.ar(Env.perc(attackTime: 0.01, releaseTime: 2, level:
1, curve: -4.0), gate: (sawtooth - 1));
```

Sedan ska bruset filtreras så att det blir ett filtersvep som förändrar frekvensinnehållet i bruset. För detta kan ett resonant lågpassfilter i `audio rate` (`RLPF.ar`) passa bra. Det är bruset som ska skickas in i filtret, och brytfrekvensen sätts till `100 + (10000 * envelope)`. Se till att brytfrekvensen aldrig går till 0Hz. Resonansen, eller `rq`, kan sättas till 0.25. Applicera sedan envelopen på ljudnivån för bruset.

```
var filteredNoise = RLPF.ar(noise, 100 + (10000 * envelope), 0.1);
var envelopedNoise = filteredNoise * envelope;
```

Avslutningsvis summeras bruset med det andra ljudet. Ibland försvinner bruset vid uppspelning då `sawtooth - 1` inte ger perfekt gate till envelopen. Då för man stoppa uppspelning och skicka över ljudet till servern igen, eller testa med 1.01 i stället.

Lyssna på ljudet och experimentera med olika inställningar. Testa att svepa ljudet uppifrån och ned i stället genom att ändra på `range` för sågtandsoscillatorn. Hur låter det?

Om man vill kan man lyxa till med både lite delay (eko) och reverb (efterklang) med följande rader i slutet

```
var reverbed = output;
3.do({ reverbed = AllpassC.ar(reverbed, 0.2, 0.2, 0.66, 1, 0)});
3.do({ reverbed = AllpassC.ar(reverbed, 0.2, { Rand(0.001,0.06) }.dup,
6)});
output = output + (reverbed * 0.5);
```

Procedurella ljudmattor

De följande två övningarna går ut på att skapa ljudmattor, eller drone-liknande musik, med procedurella metoder. Båda dessa övningar kräver en `fork`, en `loop`, en `play`, en `Splay.ar`, och en delay (`wait`) vardera.

```
(
```

```
fork{
    loop{
        play{
           Splay.ar()
        };
        1.wait;
    }
};
)
```

Detta skapar en ny parallell process (`fork`), som innehåller en loop (`loop`), vilket i sin tur innehåller information som ska spelas upp på servern (`play`), och den informationen är en array som sprids i stereofältet (`Splay`) i `audio rate` (`ar`). Detta skapas och spelas upp varje sekund (`1.wait`).

Till den första ska `Klank` användas i `audio rate`. Klank är en bank med resonatorer. Eftersom det är en bank av flera behövs flera frekvenser specificeras, samma sak för amplituden av dessa. Först ska frekvensen som används slumpas. Ett sätt att göra detta på är att ta ett slumpat heltal mellan 1 och 99 med `99.rand` och sedan multiplicera det med ett heltal mellan 1 och 10 med `(1 .. 10)`. Detta kommer att skapa olika frekvens för resonatorerna, och ett nytt sådant ljud påbörjas varje sekund.

För att resonatorerna, som egentligen är resonanta filter, ska ”sjunga” behövs en input. Denna input, är ett ljud som kommer in i filtret och får filtret att självsvänga i filtrets brytfrekvens. För att skapa ett mer procedurellt ljud ska `Crackle` användas i `audio rate` som signal in i filtret. Crackle är en kaotisk brusfunktion där ett värde strax under 1 upp till strax över 2 är användbara för att skapa ett knastrande brus.

För att ljudet ska bli större och mer intressant kör vi ljudgenereringen två gånger med `!2`.

Avslutningsvis ska ljudet passera genom en envelop som gör att ljudet sakta fadear in och ut och sedan tas den specifika synthen bort från servern. Det som sker när loopen startar en ny synth på servern är att servern sakta men säkert blir fylld av synthar som körs, och till slut tar processorkraften slut. I en envelopgenerator, och i många andra funktioner, finns något som kallas för `doneAction` vilken startar när envelopen har körts igenom. Envelopen som ska användas i detta fall är `LFGauss`, som är en funktionsgenerator som skapar en gaussisk kurva, en klockliknande kurva, som går från 0 till 1 och tillbaka till 0 över en viss specifik tid (`LFGauss.ar(duration: 1, width: 0.1, iphase: 0.0, loop: 1, doneAction: 0)`). Sätt tiden till 10 sekunder, bredden på kurvan till 25%, fasen till 0, ingen loop (0), och `doneAction` till 2.

```
LFGauss.ar(10, 0.25, 0, 0, 2)
```

Det finns olika doneAction, läs mer här:
https://doc.sccode.org/Classes/Done.html

Koden bör se ut något liknande:

```
(
fork{
    loop{
        play{
            Splay.ar({
                Klank.ar(`[99.rand * (1 .. 10)], Crackle.ar(2, 0.01));
            }!2) * LFGauss.ar(10, 0.25, 0, 0, 2)
        };
        1.wait;
    }
```

```
};
)
```

Lyssna på ljudet. Hur låter det?

_____

_____

_____

Prova att ändra `99.rand`, till att exempelvis vara `200.rand`. Prova att ändra `(2,` i Crackle till exempelvis `(1,`. Testa att ändra antalet upprepningar till `!4` osv… Hur låter det?

_____

_____

_____

Nu är det dags att skapa procedurell musik. Kopiera den tomma `fork`en från ovan. Det första som ska göras är att skapa en variabel som kan ta en slumpad frekvens. Gör detta på första raden inne i loopen men före `play`. Frekvensen kommer från en array som har fyra värden (34, 36, 41, 43). Dessa värden är egentligen inte frekvenser utan midinotnummer, och motsvarar tonerna A#, C, F och G. Läs mer här: https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies

Från arrayen med de fyra värdena ska ett väljas slumpmässigt med `.choose`, därefter ska midinotnumbret göras om till frekvens (Hz) med `midicps` (cykler per sekund). Läs mer här: https://doc.sccode.org/Classes/AbstractFunction.html

För att göra det hela lite mer intressant ska frekvensen sedan multipliceras med 2 upphöjt med 0 till 4 slumpat. Detta görs med `2 **` och `(0 .. 4).choose`. Koden bör se ut så här:

```
h = ([34, 36, 41, 43].choose.midicps) * (2 ** ((0 .. 4).choose));
```

Variabeln `h` kommer alltså att slumpat innehålla någon av tonerna A#, C, F och G i slumpvis olika oktav.

Inne i `Splay` ska själva ljudet genereras. Det görs med sinusoscillator i `audio rate` (`SinOsc.ar`). Ljudnivån på sinusoscillatorn bör sänkas till 10%, och en likadan gaussisk kurva som i det förra exemplet ska användas. Koden bör se ut så här:

```
(
fork{
    loop{
        h = ([34, 36, 41, 43].choose.midicps) * (2 ** ((0 .. 4).choose));
        play{
            Splay.ar({
                SinOsc.ar(h, 0, 0.1)}) * LFGauss.ar(10, 0.25, 0, 0, 2);
        };
        1.wait;
    };
};
)
```

Lyssna på ljudet. Hur låter det?

_____

_____

_____

För att göra det hela mer intressant ska vi upprepa ljudgenereringen 8 gånger med `!8` liknande förra exemplet. Men denna gång ska varje ljud stämmas om lite för att skapa mer intressant ljudbild. Genom att använda `exprand(min, max)` görs en exponentiell slumpning mellan min och max. Min ska i det här fallet vara `h - (h/64)` och max ska vara `h + (h/64)`. Så här:

```
SinOsc.ar(exprand(h - (h / 64), h + (h / 64)), 0, 0.1)}!8) * LFGauss.ar(10, 0.25, 0, 0, 2);
```

Lyssna på ljudet. Hur låter det och vad skiljer sig från nyss? Varför låter det annorlunda?

Prova att ändra och experimentera med antalet upprepningar (`!2` i stället för `!8`), och ändra `h/64` till exempelvis `h/16`. Ändra också delayet i loopen till exempelvis `0.25`, och prova att ändra toner som används i arrayen till `h`. Vad händer med ljudet?

Efter att ha spelat de tämligen mjuka sinustonerna ett tag, prova att ändra `SinOsc.ar` till `LFTri.ar` för att få ett ljud med lite mer övertoner. Stoppa inte uppspelningen av sinusoscillatorerna, utan ändra i koden och kör `fork`en igen. Eftersom `fork`en är en egen process så körs både sinusoscillatorerna och triangeloscillatorerna. Vad händer med ljudet?

Testa avslutningsvis med att ändra `LFTri.ar` till `LFSaw.ar` för ännu mer övertoner och harmonik. Vad händer med ljudet?