Laboration 5 – Monitorering av data

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 sonifiering för att underlätta arbetet för en operatör att monitorera en dynamisk process. Er uppgift är att skapa procedurellt ljud för den information som skickas till SuperCollider via OSC.

Funktionen i ljudet ska vara:

1. En sonifiering av kontinuerligt strömmande data (0 till 300).
2. En sonifiering av de fyra diskreta nivåerna (0, 1, 2, 3).
3. En sonifiering av de fyra olika varningsmeddelandena som skickas. "Risk for overheating", "Bipolar capacity low", "Unstable output", "Zero target null"

Ni bestämmer vilken sonifieringsapproach ni vill använda. Ni kan till exempel använda parametermappad sonifiering för att representera strömmande data, auditory icons för de olika nivåerna, och earcons för meddelandena.

Inför laborationen

Förberedelse – 1. Ladda ner ”gränssnittet”

Ladda ner och installera Processing samt gränssnittet. Gränssnittet är inte vackert. Det är faktiskt med flit gjort för att se rörigt och klantigt ut. Ni vet, ungefär som när en ingenjör som vet vad allt betyder och innebär, ”designar” ett gränssnitt för vanliga människor. Det hela är gjort i Processing som är en programmeringsmiljö som lämpar sig bra för grafiska saker. Gränssnittet skickar all data över OSC, över följande adresser ”level”, ”data”, och ”error”. Och data skickas över NetAddr(127.0.0.1, 57120).

Läs igenom denna artikel från Interactive Sonification Workshop 2019 som handlar om sonifiering av monitorering av dynamiska processer, ISon2019-01-Rönnberg.pdf

Förberedelse – 2. Bekanta er med SuperCollider-koden

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

Synthdefinitioner

Koden har en kort/liten synthdefinition (`dataSonification`) som från början innehåller en triangelvåg (`LFTri.ar`) som tar emot ett inputargument, `freq`.

Klientsideskript

På klientsidan stoppas synthdefinitionen på servern in i `~dataSynth`, och därefter följer ett gäng OSC-lyssnare och därefter en funktion som mappar input-data till frekvens. Denna funktion behövs kanske inte egentligen, men jag tänkte att det kunde vara ett bra tillfälle att visa hur data kan skickas till och returneras från en funktion i SuperCollider.

Nivålyssnaren

Den första OSC-lyssnaren lyssnar på adress `level`. I det här fallet görs inget annat i den här lyssnaren, utan ni får fundera på vad som ska sonifieras och hur det ska sonifieras. If-satsen används för att bara sonifiera när statusen ändras.

Datalyssnaren

Den andra OSC-lyssnaren lyssnar på värden som kommer adresserade som `data`. OSC-meddelandet som kommer på position 0 (`inputmsg[0]`) är en sträng, `data`. Själva datavärdet finns på position 1. För att inte skicka data i onödan till synthdefinitionen använder jag environmentvariabeln `~currentData` och inkommande datavärde måste vara annorlunda för att någon data ska skickas till synthdefinitionen.

Det datavärde som kommer via OSC görs om från en sträng till en `int` med `.asInteger`. Sedan skickas heltalet till funktionen `~mapPitch` med hjälp av `.value()`. Det är denna externa funktion som är lite onödig, men ni kan väl se det som god programmeringsvana att använda många små funktioner för olika ändamål i stället för att stoppa in massor med funktionalitet i en funktion.

Avslutningsvis skickas frekvensen till `\freq` på synthdefinitionen.

Felmeddelandelyssnaren

Den tredje OSC-lyssnaren lyssnar på adress `error`. Det inkommande värdet skrivs ut i post-fönstret med hjälp av `.postln`. Annars görs inget annat i den här lyssnaren, utan ni får fundera på vad som ska sonifieras och hur det ska sonifieras.

Den externa funktionen

Sist kommer den externa funktionen, `~mapPitch`, som tar emot ett inputargument, `freq`. I den här funktionen används en linjär mappning av inputvärdet, som är mellan 0 och 300, till frekvens, mellan 220Hz och 880Hz (dvs två oktaver). Det mappade värdet stoppas i en variabel, `frequency`, vilket sedan returneras med `frequency.value` sist i funktionen.

Skapa kopplingar mellan data och ljud

Kolla igenom de tidigare laborationshandledningarna och använd den kunskapen när ni jobbar med denna laboration. Här följer dock en del tankar och reflektioner som förhoppningsvis kan komma till användning.

Parametrisk sonifiering - tonhöjd

Dataströmmen skulle kunna sonifieras med så kallad parametermappad sonifiering. Det innebär att ett grundljud designas och sedan mappas vissa ljudelement till datavärden, så att ljudet förändras i relation till inkommande data. I grundkoden görs detta genom att koppla datavärde 0 till 220Hz och datavärde 300 till 880Hz. Detta gör att frekvensen/tonhöjden på sonifieringen varierar i relation till data, men också att det blir falskt när vissa datavärden inte perfekt passar en ton i vår skala. Detta skulle kunna åtgärdas genom att i stället för frekvens i Hz koppla data till MIDInotnummer vilka sedan görs om till frekvens med `midicps`.

Läs mer om MIDInotnummer här
https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies

Ett sätt att göra det hela mer musikaliskt på skulle kunna vara att kvantisera MIDInotnummer till specifika toner i en skala som ni väljer. I följande kod har jag valt 5 toner i en array, `~originalTones`, som jag lägger till i en annan array, `~quantTones`, och sedan

höjer jag tonerna i `~originalTones` med en oktav (12 tonsteg eftersom det är hel- och halvtoner), och så lägger jag till totalt 8 oktaver upp.

```
// C, Eb, F, G, Bb
~originalTones = [12, 15, 17, 19, 22];
~quantTones = ~originalTones;
for (0, 7) { arg index;
   ~originalTones = ~originalTones + 12;
   ~quantTones = ~quantTones.addAll(~originalTones);
};
```

Därefter kan följande kod användas för att mappa inkommande frekvens till närmsta ton i vår kvantiserade skala och sedan returnera frekvensen.

```
~mapTones = { arg freq;
   var incomingNote = freq.cpsmidi;
   var comparisonValue;
   var notePosition;
   var mappedFreq;
   for (0, ~quantTones.size-1){ arg index;
      var tempValue = (~quantTones[index] - incomingNote).abs;
      if (comparisonValue.isNumber == false) {
         comparisonValue = tempValue;
      };
      if (tempValue < comparisonValue) {
         comparisonValue = tempValue;
         notePosition = index;
      };
   };
   mappedFreq = ~quantTones[notePosition].midicps;
   mappedFreq.value;
};
```

Men, det kan vara värt att tänka på att när datavärdena justeras till korrekta MIDInotnummer i stället för frekvens så tappar vi upplösning i representationen av data. Och, används sedan en kvantiserad skala med bara några toner i, kan ännu mer precision förloras.

Parametrisk sonifiering - harmoni

Ett intressant sätt att använda parametermappad sonifiering är genom att använda ett ljud som består av två oscillatorer i samma frekvens. När datavärdena ändras så ändras frekvensen på dessa två oscillatorer lite lite åt olika håll. Ju högre datavärdena blir desto högre blir `detune`, då blir ljudet falskare och falskare ju högre värdena blir.

```
var sine1 = SinOsc.ar(220 + detune);
var sine2 = SinOsc.ar(220 - detune);
```

Effekten som uppstår när två frekvenser ligger nära varandra, men inte helt perfekt i harmoni kallas för beating. När de två ljuden är i perfekt frekvens så blir ljudet starkare, men det låter som ett ljud. När ljuden börjar separeras i frekvens uppstår en sakta rytm, denna rytm ökar i tempo när frekvenserna separeras mer, och efter ett tag är det inte ett ljud med en rytm utan det hörs att det är två olika toner. Det innebär att när man justerar frekvens så här, får man två ljudeffekter på köpet.

Parametrisk sonifiering - ljudvolym

Ljudvolymen på ljudet kan också justeras i relation till datavärden. Det är dock viktigt att tänka på hörbarheten, dvs att 0 eller 1 i datavärde inte representeras med 0 eller 0.01 i ljudvolym så att låga datavärden helt enkelt inte hörs.

Parametrisk sonifiering - klangfärg

Det finns många olika sätt att justera klangfärgen i ett ljud. Till exempel kan pulsbredden på en fyrkantsvågform justeras mellan 0 och 0.5, eller mellan 0.5 och 1. Kom ihåg att vi inte hör skillnad i fas, så 0.25 och 0.75 låter lika även om vågformerna ser olika ut.

Det går också att mixa olika vågformer med varandra, som exempelvis en sinus och en sågtand. Detta löses enkelt med ett inputargument, exempelvis `waveMix`, som varieras mellan 0 och 1 i relation till datavärdet. Kom ihåg att SuperCollider läser matte från vänster till höger även om det gör ont i ögonen när man ser koden.

```
var output = (1 - waveMix * sinus) + (waveMix * sawtooth);
```

Ett annat sätt att justera klangfärgen är genom att använda distorsion. Om en sinusvåg distas förändras den till att bli mer och mer av en fyrkantsvåg. Detta kan till exempel göras med hjälp av `Clip.ar(input, lowClip, highClip)`. Där `lowClip` sätter värdet som ljudet klipps på den negativa sidan av vågformen och `highClip` för den positiva sidan. Eftersom ljudet klipps av, blir det svagare, och behöver förstärkas proportionerligt för att låta bra.

Datavärden går också att använda för att styra brytfrekvens på ett filter (LPF, BPF, eller HPF), så att sonifieringsljudet har högre frekvenser för höga datavärden och lägre frekvensinnehåll vid låga värden.

Parametrisk sonifiering - tidsaspekter

Olika tidsaspekter går också att koppla till datavärden. Exempelvis kan en fyrkantsvåg som svänger mellan 0 och 1 (justeras med `range`) användas för att stänga av och sätta på ljudet från en annan oscillator. Frekvensen på fyrkantsvågen kan sedan kopplas till data med exempelvis snabbare frekvens för högra datavärden.

En annan tidsaspekt är de olika inställningarna i en envelopgenerator, som attack, decay och release. Där exempelvis releasetiden kan mappas till datavärde med längre utklingningstider, releasetider, för högre datavärden.

Parametrisk sonifiering - åt vilket håll?

En sak som är viktig att tänka på är åt vilket håll som sonifieringen ska ändras åt i relation till data. Ska tonhöjden öka när datavärdena ökar? Det funkar i många fall som kanske temperatur eller frekvens, men exempelvis inte för vikt. Vid vikt tolkar vi ofta en låg frekvens som en högre vikt jämfört mot ett ljud i en högre frekvens.

Ljudikoner och earcons

En auditory icon, eller ljudikon är ett kort ljud som ofta har en naturlig koppling mellan ljudet och den händelse som den är kopplad till. Tänk till exempel på när papperskorgen i datorn töms och ljudet av ett papper som knycklas ihop. Ni skulle kunna spela in ljud som ni kan använda för att symbolisera olika event i inkommande data, och sedan procedurellt manipulera dessa i relation till de olika eventen. Ni kan också helt procedurellt framställa dessa ljudikoner och ha full kontroll över alla ljudparametrar.

Earcons är korta, ofta mer musikaliska, sammansatta ljud. Tänk er korta signaturmelodier på någon sekund som används för att förtydliga skeenden i en monitoreringsprocess. Det skulle kunna vara så enkelt som ett särskilt pling, eller några toner som tillsammans bilder en enhet. Start- och stängljuden på en dator är bra exempel på detta. I det här fallet finns ingen naturlig koppling mellan ljudet och eventet som för ljudikoner, utan det är snarare ljud som vi har bestämt ska låta på ett visst sätt och då representera något särskilt. Här kan det vara

bra att kolla upp exemplen om `fork` eller den enkla sequencern som användes till trumljuden.