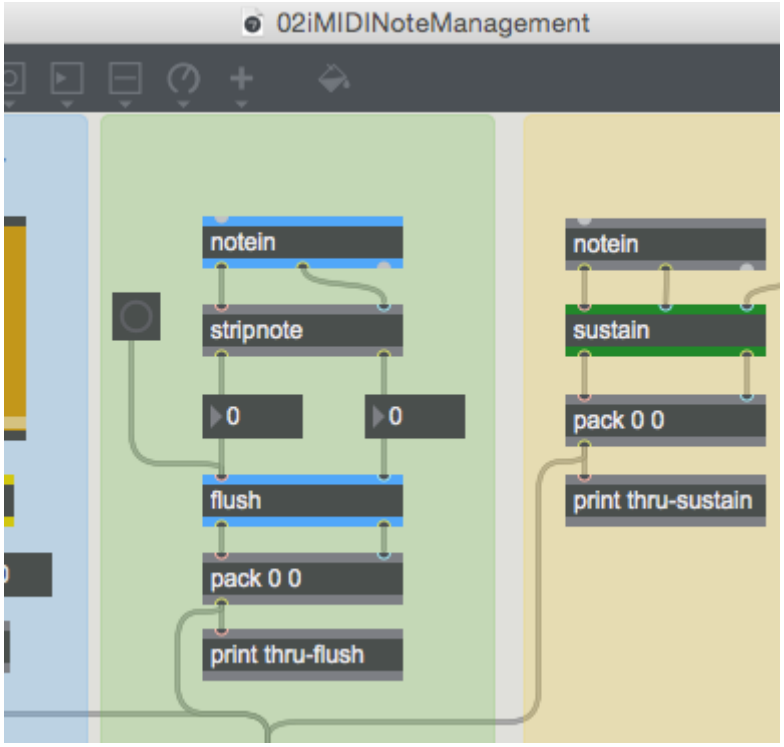
Kontrolloppgave til øvelsen *Note Management*

Denne øvelsen går videre i bruk av MIDI-protokollen. Viktige funksjoner som *stripnote* og *flush* blir gjennomgått. Til slutt skal vi gjøre en oppgave som kombinerer denne MIDI-øvelsen med tidligere øvelser, f.eks. de som omhandlet tilfeldigheter.

Gå gjennom øvelsen *Note Management* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og les gjennom det som er uklart en gang til. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 22

Lag en prosedyre som spiller tilfeldige MIDI-toner.

22.1 Tonene skal ha et omfang på åtte oktaver, og laveste MIDI-note skal være MIDI-notenummer 24.

22.2 Anslagsstyrken *(velocity)* skal være en tallrekke på 12 verdier som oscillerer frem og tilbake mellom verdiene 60 og 120.

22.3 Varighetene skal være en tallrekke på 12 verdier som går i oppadstigende rekke mellom 10 og 1210 millisekunder.

22.4 Hver nye note trigges ved å trykke på en knapp.

Nye objekter introdusert i denne øvelsen:

sustain

flush

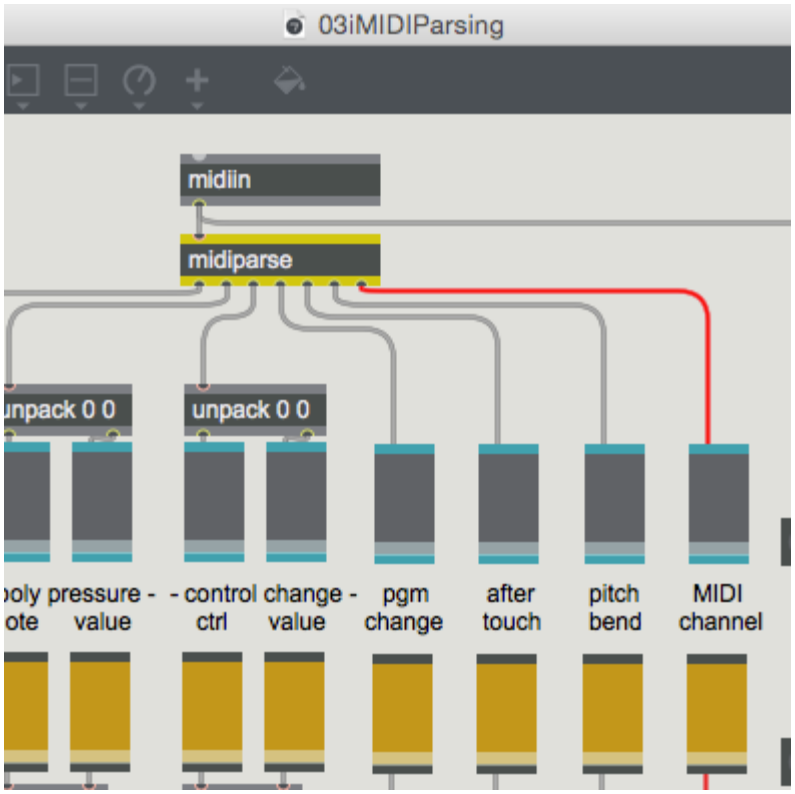
stripnote

makenote

Kontrolloppgave til øvelsen *Parsing*

Denne øvelsen går hovedsaklig igjennom objektene *midiparse* og *midiformat* som er nyttige alt-i-ett- objekter for behandling av MIDI-data. På grunn av disse objektenes natur vil dette være en øvelse med mindre praktiske oppgaver enn vanlig. Det er allikevel viktig å forstå disse objektene siden de kan være nyttige i håndteringen av kompliserte MIDI oppsett.

Gå gjennom øvelsen *Parsing* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 23

I denne øvelsen har vi brukt blant annet objektene *midiparse*, *midiformat*, *midiin* og *midiout*. Benytt disse objektene til å lage en enkel patch der du kan spille toner fra midi-keyboardet. I patchen skal du også kunne bruke pitchbend hjulet til å «bøye» tonene på vanlig måte. Midi note number, velocity og pitchbend verdier skal vises med slidere.

Nye objekter introdusert i denne øvelsen:

speedlim

xbendin

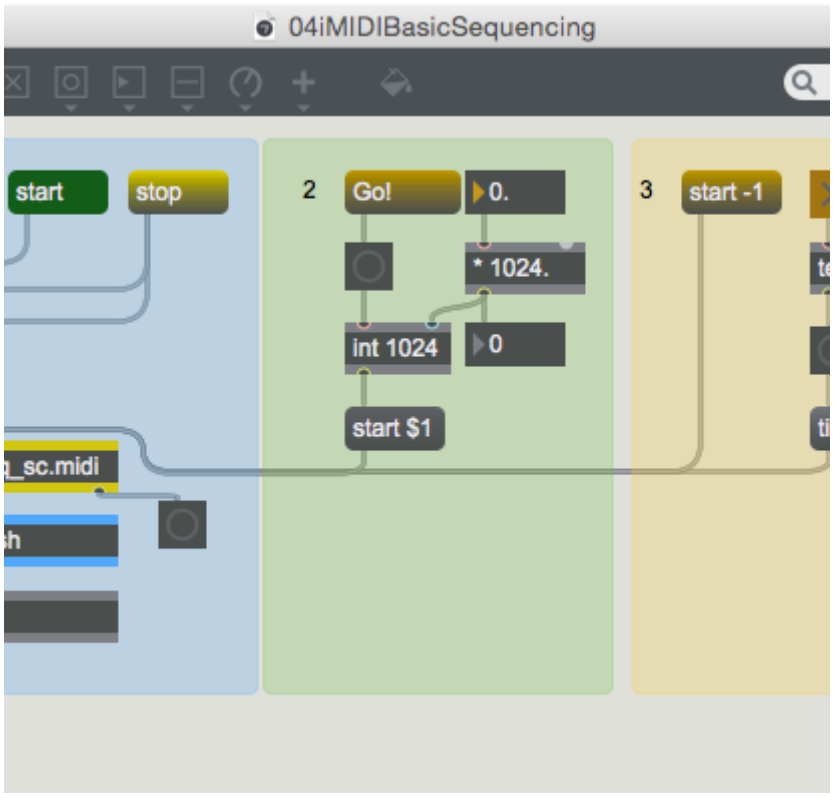
midiformat

midiparse

Kontrolloppgave til øvelsen *Basic Sequencing*

Denne øvelsen går igjennom enkel grunnleggende MIDI-innspilling med objektet *seq*. Legg merke til bruken av *midiflush* og den fleksible tempoangivelsen.

Gå gjennom øvelsen *Basic Sequencing* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 24

Lag en patch med to deler der den første delen spiller av en sekvens *(sequence)* i halvt tempo og den andre delen spiller av en sekvens som går fra et tempo på 50 bpm til 150 bpm i løpet av 5 sekunder.

Nye objekter introdusert i denne øvelsen:

tempo

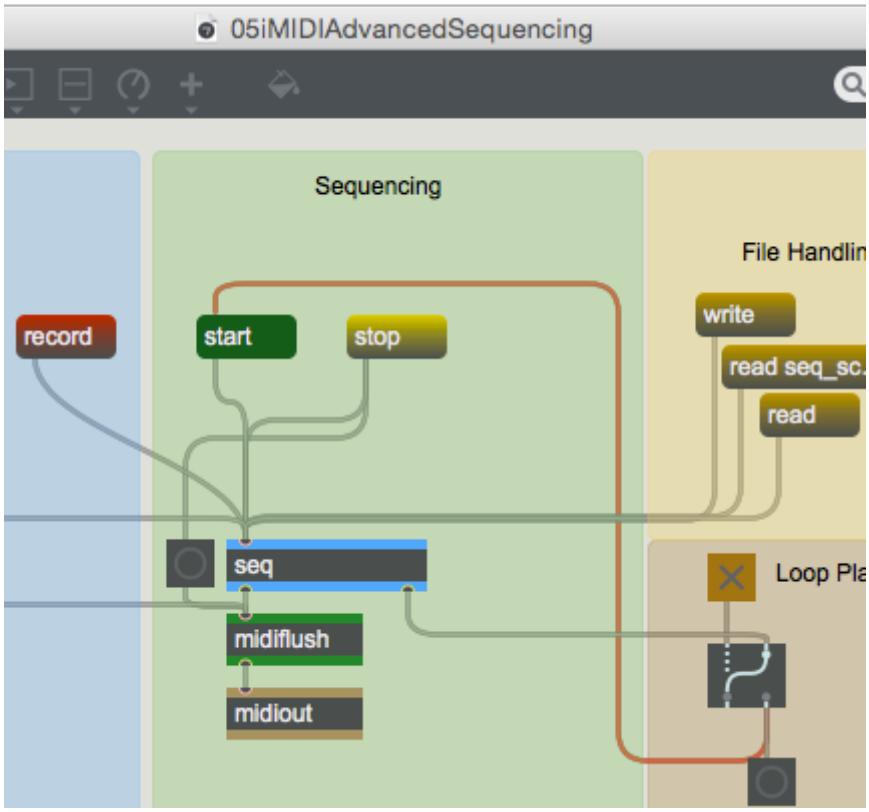
midiflush

seq

Kontrolloppgave til øvelsen *Advanced Sequencing*

Vi fortsetter med MIDI inn- og avspilling og legger til muligheten for å lagre en MIDI-fil sammen med patchen.

Gå gjennom øvelsen *Advanced Sequencing* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 25

Spill inn en MIDI-fil og kall den «midifile01». Spill inn en MIDI-fil til og kall den «midifile02».

25.1 Når patchen åpner skal «midifile01» automatisk lastes inn.

25.2 Når du trykker på en knapp skal «midifile02» lastes inn.

25.3 Patchen skal være utstyrt med vanlige kontroller for inn- og avspilling.

25.4 Avspillingen skal ha mulighet for å spille musikken i løkke *(looping)*.

Nye objekter introdusert i denne øvelsen:

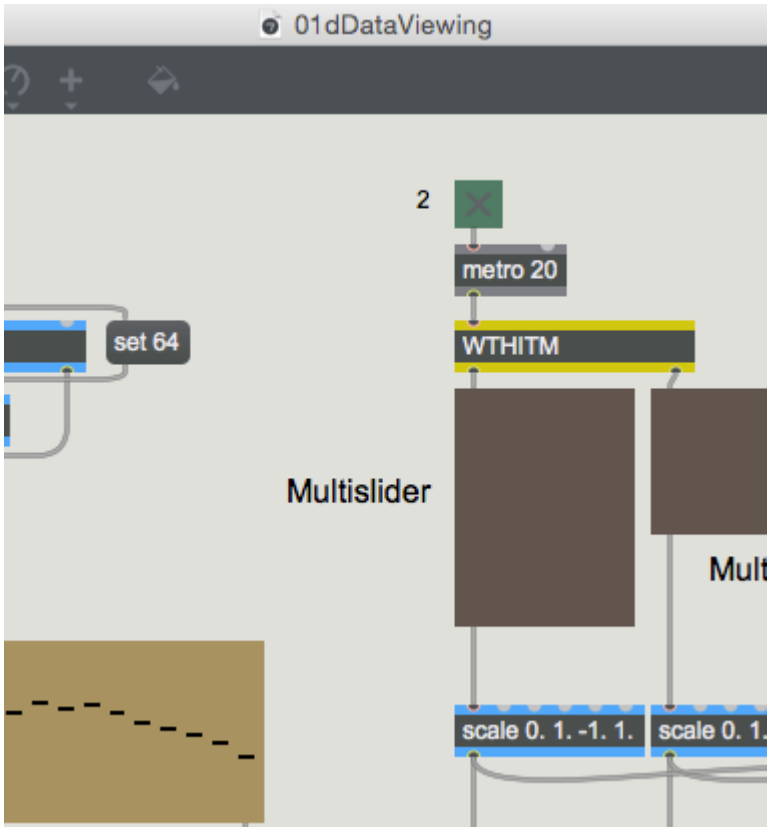
seq

midiflush

Kontrolloppgave til øvelsen *Data Viewing*

Her er vi tilbake til en litt mer kompleks oppgave etter noen enkle MIDI-øvelser. Det er først og fremst objektet *multislider* som er sentral i denne oppgaven. Vi skal også gå igjennom funksjonen *Watchpoints*.

Gå gjennom øvelsen *Data Viewing* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 26

26.1 Lag en patch som repeterer en rekke toner som er lagret i et *multislider*-objekt. Dette er det samme objektet som vises i eksempelpatchens del 1. For å gjøre det litt mer fleksibelt, sett inn et *number*-objekt som velger hvor lang løkken (loopen) skal være. Dette nummerobjektet skal også velge hvor mange toner som finnes i *multislider*-objektet og hvor mange toner som vises i *multislider*-objektet. For å løse dette kan du bl.a. se på objektene *counter*, *uzi*, *multislider* og *peak*. Se også på *size*-beskjeden til *multislider*-objektet.

26.2 Lag en prosedyre som følger musas bevegelse over skjermen og som i tillegg er garantert å fungere på alle datamaskiner ved å måle den enkelte datamaskins skjermstørrelse:

Når musa beveger seg horisontalt fra venstre til høyre på skjermen, så forandrer du omfanget til MIDI-notenummerne som kommer ut av *multislider*-objektet fra null til to oktaver.

Når musen beveger seg vertikalt på skjermen forandrer du anslagsstyrken (velocity) på MIDI-notene fra ingenting til maksimum.

Bruk en kombinasjon av objektene *change, button* og *flush* for å automatisk skru av noter som henger seg opp når du beveger på musa. Disse forandringene i anslagsstyrke (velocity) og omfanget til MIDI-notenummerne, skal du gjøre på resultatet av det du gjorde i 26.1, den første delen av denne oppgaven.

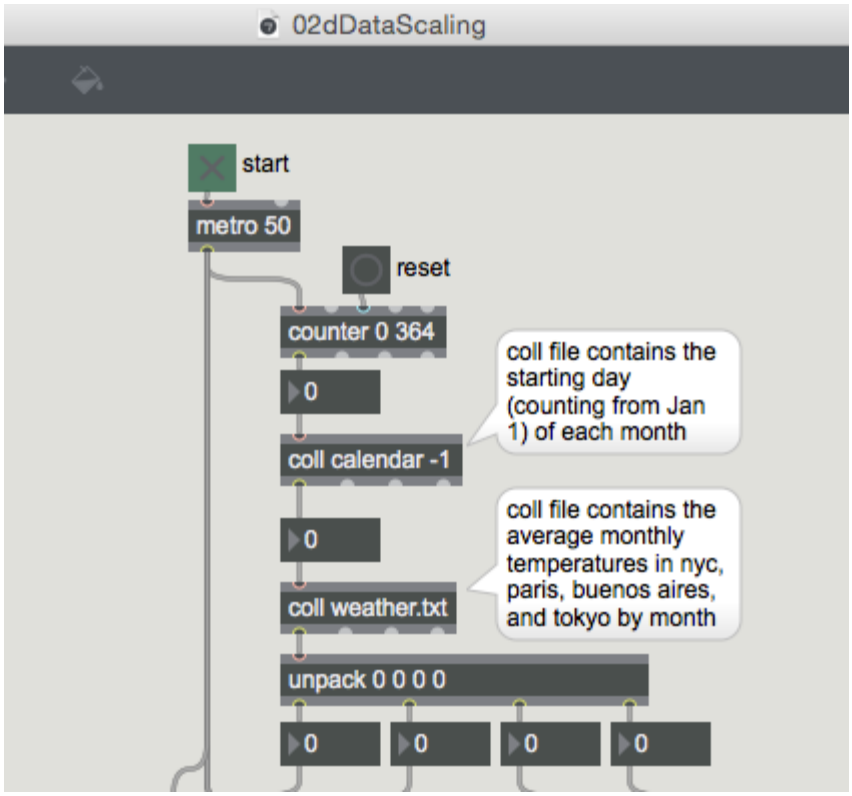
Nye objekter introdusert i denne øvelsen:

multislider

Kontrolloppgave til øvelsen *Data Scaling*

Denne øvelsen ser på en rekke objekter for skalering og lagring av data. Et viktig objekt er *coll. coll* er et av de vanligste objektene for å lagre ulike typer data i Max, enten det er symboler eller tallrekker.

Gå gjennom øvelsen *Data Scaling* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 27

27.1 Lag en prosedyre som lager tre parallelle strømmer med tilfeldige tall.

27.2 Disse tallstrømmene skal glattes ut slik at ikke det blir for store hopp mellom verdiene.

27.3 De tre tallstrømmene skal vises i et *multislider*-objekt.

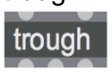
27.4 Minimum- og maksimumsverdi for *multislider*-objektet skal settes automatisk basert på minimum- og maksimumsverdiene i de tre tallstrømmene

Nye objekter introdusert i denne øvelsen:

iter

peak

trough



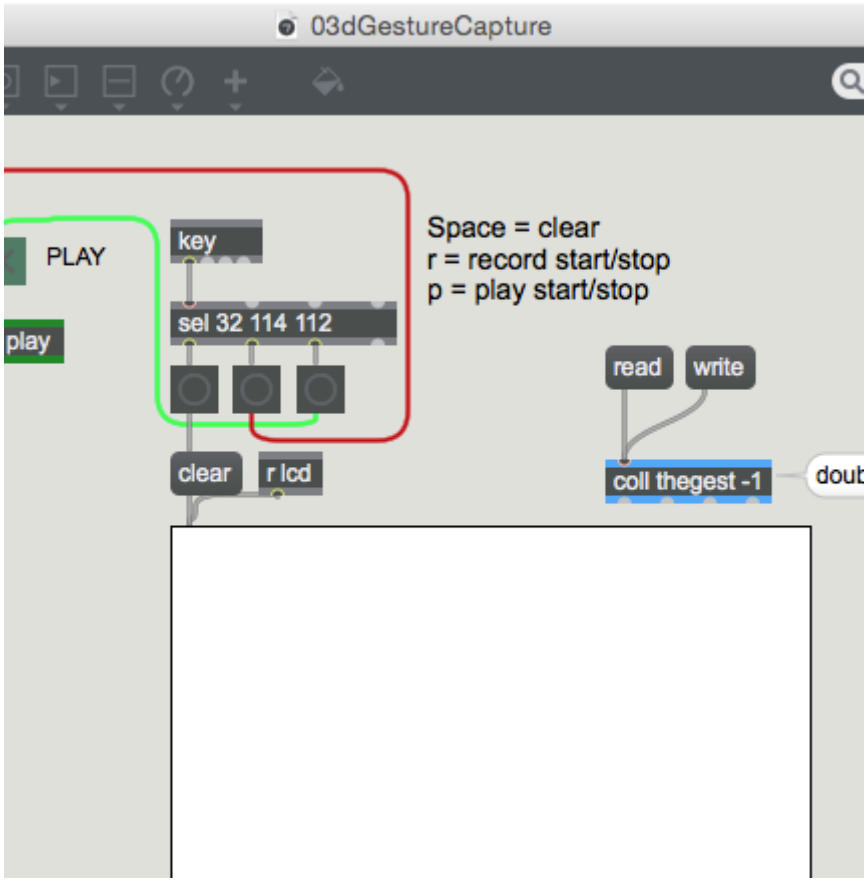
slide

Kontrolloppgave til øvelsen *Gesture Capture*

I denne øvelsen arbeider vi med mer kompleks bruk av *coll*-objektet. Innspilling av ulike former for data er viktig, og *coll*-objektet kan f.eks. brukes til å spille inn gester slik som i dette eksempelet. I tillegg vil vi jobbe med oppløsning, både når det gjelder hvor ofte data blir aksessert og hvor nøyaktig dataene blir representert. To nøkkelord for oppløsning er samplefrekvens (*sampling rate)* og bitdybde (*bit depth)*. Samplefrekvensen sier noe om hvor ofte vi måler en verdi (altså hvor stor oppløsning vi har i tid) mens bitdybdensier noe om hvor stor nøyaktighet vi har når vi måler. MIDI har f.eks. en oppløsning på 7 bit, dvs 128 verdier, og når man arbeider med sensorer er oppløsningen viktig.

Gå gjennom øvelsen *Gesture Capture* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet. Vær oppmerksom på en ting som kan være litt forvirrende med øvelsespatchen. Musens bevegelse over ***hele*** dataskjermen representeres av det lille hvite feltet til *lcd*-objektet. Du skal altså *ikke* trykke inne i *lcd*-objektet men bare bevege musen over skjermen etter at du har satt i gang opptaket av gesten.

Kommandoen *paintpoly* gjør at gesten blir tegnet inn i *lcd*-objektet. Denne kommandoen gjør det mulig å bl.a. tegne strektykkelsen ut fra hvor fort man beveger musen. Les dokumentasjonen til *lcd*-objektet for å forstå hvordan *paintpoly* fungerer.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 28

Ta utgangspunkt i patchen som følger med i eksempelet, og gjør følgende forandringer:

28.1 Sett inn muligheten til å spille inn gestene med forskjellige oppløsninger (sample rate).

28.2 Sett inn muligheten for å spille av gestene i ulike hastigheter.

Viktig objekt:

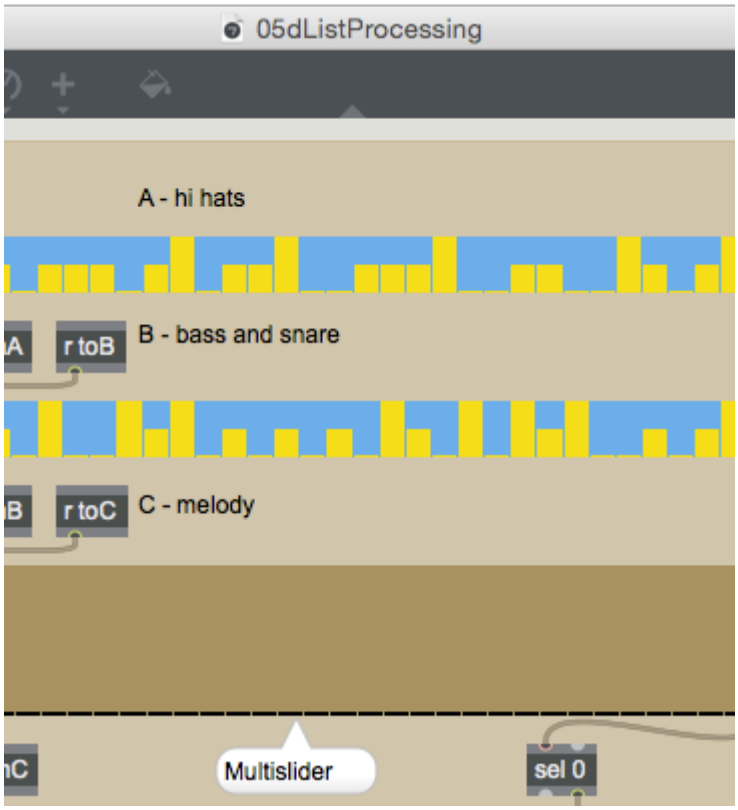
coll

Kontrolloppgave til øvelsen *List Processing*

Øvelsen *Cellblock* benytter seg av *jit.cellblock*-objektet som gir et visuelt grensesnitt til data. Dette objektet kan være nyttig for visning og redigering av verdier i patchen men i denne omgangen behøver vi ikke dette. Vi hopper derfor rett til øvelsen *List Processing*.

Øvelsen *List Processing* er en lang og omfattende øvelse fordi den dekker et viktig tema; listeprosessering. Vi skal gå gjennom step-sequensere, sannsynlighetsberegning og ulike former for listeprosessering. Sannsynlighetsberegning er et viktig område av informasjonsteknologien og er sentralt i alt fra aksjehandel og beregning av oljepriser til spesialeffekter i Hollywoodfilmer. Siden denne øvelsen er såpass omfattende har vi delt den inn i fire underdeler.

Gå gjennom øvelsen *List Processing* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 29, del 1: Step sequenser

29.1.1 Ta utgangspunkt i step-sequenseren fra øvelsen. Forandre den slik at den ikke har 32 skritt, men maksimum 8.

29.1.2 For hver ny runde step-sequenceren går gjennom verdiene skal den velge en tilfeldig lengde på sekvensen fra 1 til 8 skritt. Den tilfeldige lengden skal være forskjellig i patchens tre spor (hi hat, bass/snare, melody).

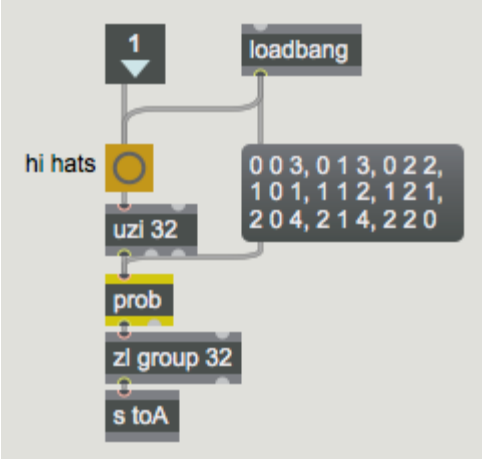
29.1.3 Husk at siden sekvensen har en maksimumslengde på 8 så må også multisliderene skifte antall slidere til 8.

29.1.4 Ved å trykke på en knapp skal man kunne reversere rekkefølgen på alle verdiene.

Oppgave 29, del 2: Sannsynlighet

prob-objektet brukes til å gi ut tallrekker basert på sannsynlighetsmønstre som angis av brukeren. Om vi går nederst til venstre i øvelsespatchen ser vi et område med overskriften generate new patterns. Delen hi hats genererer data som spilles av hihaten. Disse dataene (eller det musikalske materialet om du vil) genereres ut ifra en sannsynlighetsvekting ved hjelp av *prob*-objektet.

Følgende liste er koblet til *prob*-objektet: 0 0 3, 0 1 3, 0 2 2, 1 0 1, 1 1 2, 1 2 1, 2 0 4, 2 1 4, 2 2 0



Denne listen er oppdelt i tre og tre tall. De to første tallene beskriver en tilstandsovergang, mens det siste tallet beskriver en sannsynlighetsvekt. Tilstand er bare et annet ord for tallet som *prob*-objektet gir ut. Vi bruker de tre første tallgruppene over som eksempel:

0 0 3: Overgang fra tilstand 0 til tilstand 0 (ingen endring), sannsynlighetsvekt 3.
0 1 3: Overgang fra tilstand 0 til tilstand 1, sannsynlighetsvekt 3.
0 2 2: Overgang fra tilstand 0 til tilstand 2, sannsynlighetsvekt 2.

Sannsynlighetsvekten sier noe om hvor sannsynlig det er at *prob*-objektet går fra en gitt tilstand til en annen. Jo større vekten er relativt til de andre vektene, jo mer sannsynlig er det at den tilhørende tilstandsovergangen vil skje.

Først når vi har sett eller oppgitt alle sannsynlighetsvektene for en gitt starttilstand kan vi regne ut den faktiske sannsynligheten for at en viss overgang skal skje. Dette gjør vi ved å først summere vektene. Vi bruker igjen eksempelet over, og siden 0, 1 og 2 er de eneste tilstandene som brukes har vi oppgitt alle de tre mulighetene som finnes for en overgang fra tilstand 0. Vi summerer så alle vektene: 3 + 3 + 2 = 8. Dette tallet dividerer vi hver vekt med for å få sannsynligheten (et tall fra 0 til 1) for at hver overgang skal skje. Ganges dette tallet med 100 vil man få sannsynligheten i prosent. Igjen fra eksempelet over:

0 0 3: 3/8 = 0.375 = 37.5%. Det er 37,5% sjanse for at *prob*-objektet vil gi ut en 0 om den forrige verdien den gav ut også var 0.

0 1 3: 3/8 = 0.375 = 37.5%. Det er 37,5% sjanse for at *prob*-objektet vil gi ut en 1 om den forrige verdien den gav ut var 0.

0 2 2: 2/8 = 0.25 = 25%. Det er 25% sjanse for at *prob*-objektet vil gi ut en 2 om den forrige verdien den gav ut var 0.

Merk at alle disse sannsynlighetene summert sammen (37.5% + 37.5% + 25%) blir 100%, som igjen bekrefter at alle overgangsmuligheter er dekket opp.
De andre tallgruppene i lista som er koblet til *prob*-objektet beskriver det samme konseptet, men for starttilstandene 1 og 2.

La oss se på et eksempel hvor vi går andre veien. Vi har tilstandene 0 og 1, og ønsker at *«prob»*-objektet skal oppføre seg slik:

Om nåværende tilstand er 0, skal det være lik sannsynlighet for at neste tilstand er 0 eller 1, altså 50% sjanse for begge.

Om nåværende tilstand er 1, skal det være 20% sannsynlighet for at neste tilstand er 0, og 80% sannsynlighet for at neste tilstand forblir 1.

Merk at total sannsynlighet for begge starttilstandene er akkurat 100%. Om dette ikke er tilfelle har vi angitt feil sannsynligheter.

Når vi så skal lage tabellen vi vil gi til *prob*-objektet, må vi gjøre om disse sannsynlighetene til sannsynlighetskoeffisienter, noe som gjøres lettest ved å regne dem om til brøker. Dividenden (det øverste tallet) vil da være sannsynlighetskoeffisienten vi gir videre til *prob*-objektet. For de tre forskjellige sannsynlighetene vi har blir det:

50% = 0.5 = 1/2

20% = 0.2 = 1/5

80% = 0.8 = 4/5

overgang	vekting	vekting i prosent
0-0	1	50
0-1	1	50
1-0	1	20
1-1	4	80

Å gjøre dette om til en beskjed som *prob*-objektet tar imot er enkelt:
0 0 1, 0 1 1, 1 0 1, 1 1 4

Ikke la deg narre av de halvveis abstrakte konseptene, det at *prob*-objektet benytter seg av tall som tilstander betyr ingenting for oss, siden vi kan tolke tallene *prob*-objektet gir ut akkurat som vi vil. Vi kan for eksempel si at 0 vil trigge en basstromme, mens 1 vil trigge skarptromme. Da vil hele tabellsystemet enkelt og greit bare beskrive hvor sannsynlig det er at en skarptromme vil komme etter en basstromme, eller omvendt.

Om noe virker uklart bør man som alltid lese hjelpefilen og referansefilen til *prob*-objektet. Ut ifra dette dokumentasjonsmaterialet burde det være mulig å få en forståelse av hvordan *prob*-objektet fungerer.

Sannsynlighetsoppgave:
29.2 Lag en rutine som skriver 25 verdier. Tallene skrives til Maxvinduet med følgende fordeling:

overgang	vektig i prosent
0-0	33,33

0-1	33,33
0-2	33,33
1-0	25
1-1	25
1-2	50
2-0	100
2-1	0
2-2	0

Oppgave 29, del 3 *zl*-objektet

29.3.1 Ved hjelp av *zl*-objektet, lag en prosedyre som bruker objektene *uzi* og *urn* til å generere 25 tall mellom 0 og 24.

29.3.2 Gjør dette om til en liste og skriv det ut i Maxvinduet.

29.3.3 Sorter listen i stigende rekkefølge og skriv det ut i Maxvinduet.

29.3.4 Roter listen med tre tall og skriv det ut i Maxvinduet.

29.3.5 Del opp listen i puljer på fem tall, skriv hver pulje baklengs og skriv det ut i Maxvinduet.

29.3.6 Skriv ut siste tall i hver pulje.

Oppgave 29, del 4 *vexpr*-objektet

Du husker kanskje at vi gikk gjennom objektet *expr* i øvelsen *Designing Equations*. Det finnes et tilsvarende objekt for matematiske ligninger på lister. Objektet heter *vexpr*.
Ved hjelp av *vexpr*-objektet, gjør følgende tre korte øvelser:

29.4.1 Ta de to listene 10 20 30 40 50 60 og 1 2 3 4 5 og summer de. Skriv de ut til Max-vinduet.

29.4.2 Bruk listen 0 10 20 30 40 50 60 70 80 90. Hver gang du trykker på en knapp skal hver verdi forandre seg tilfeldig med en maksimumsforandring på +/- 2. Skriv resultatet ut til en *multislider*.

29.4.3 Bruk den samme listen 0 10 20 30 40 50 60 70 80 90. Avhengig av hvilken knapp du trykker på skal hver verdi øke eller synke med +/- 5. Skriv resultatet ut til en *multislider.*

Nye objekter introdusert i denne øvelsen:

vexpr zl prob

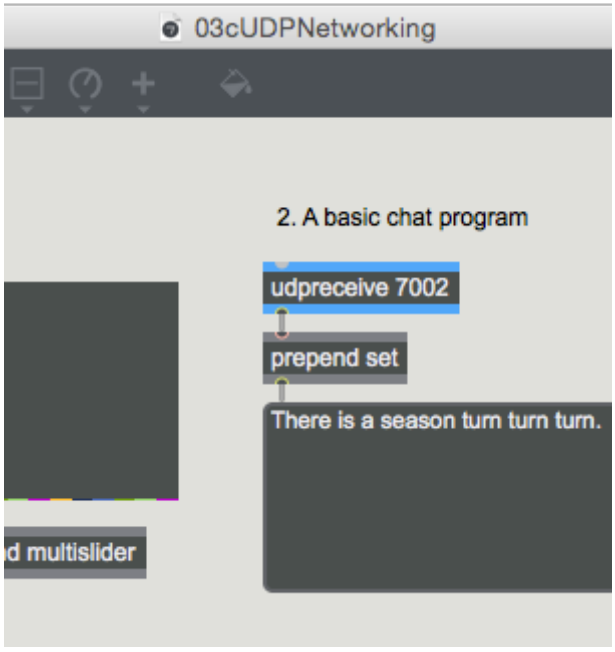
Kontrolloppgave til øvelsen *UDP Networking*

Vi hopper rett til øvelsen *UDP Networking*. I øvelsene *Human-Interface Devices* og *Serial Communication* kreves det maskinvare for å gjennomføre øvelsene. Vi trenger en spillkontroller (joystick eller lignende) for å benytte HID-protokollen i øvelsen *Human-Interface Devices* og et serielt grensesnitt som for eksempel et Arduinokort eller lignende for å bruke den serielle protokollen i øvelsen *Serial Communication*.

Har du tilgang til slike maskinvarekontrollere så anbefales det på det sterkeste å gå gjennom disse øvelsene. Bruk av ekstern maskinvare og sensorer er sentralt i arbeidet med Max. Hvis ikke så kan du hoppe over disse øvelsene og heller gå tilbake til dem når du eventuelt har tilgang til slik maskinvare.

I øvelsen *UDP Networking*skal vi se på hvordan vi kan bruke Max til å kommunisere via nettet. Det kan være mange grunner til gjøre dette. Noen prosjekter kan være så prosessorkrevende at man trenger flere maskiner koblet sammen for å utføre dem. Andre prosjekter kan være basert på at man kommuniserer mellom ulike maskiner plassert på ulike steder. Uansett hva målet er så kan nettverksbaserte patcher være en enkel måte å løse ulike problemer på.

Gå gjennom øvelsen *UDP Networking* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 30

Her følger tre deloppgaver. Siden vi ikke kan forvente at du disponerer to datamaskiner i et nettverk, vil vi løse disse oppgavene med hjelp av argumentet «localhost». Vi anbefaler allikevel at du lærer deg hvordan nettverkskommunikasjon mellom datamaskiner fungerer siden dette er nyttig i mange sammenhenger.

30.1 Lag en prosedyre som bruker en *multislider* til å sende informasjon. *Multislideren* skal ha 10 slidere, 4 alternerende farger, kontinuerlig data ut når man bruker musen, og *slider style* skal være satt til *bar*.

Tre slike multislidere skal sende data til tre andre multislidere via argumentet «localhost».

30.2 Lag en prosedyre som muliggjør chatting mellom to ulike deler av patchen ved hjelp av argumentet «localhost».

30.3 Lag en prosedyre som muliggjør sending av MIDI-note data mellom to ulike deler av patchen ved hjelp av argumentet «localhost». Den ene delen av patchen skal sende MIDI-note data fra et MIDI-keyboard mens den andre delen av patchen skal spille dette av som MIDI-noter.

Nye objekter introdusert i denne øvelsen:

Textedit

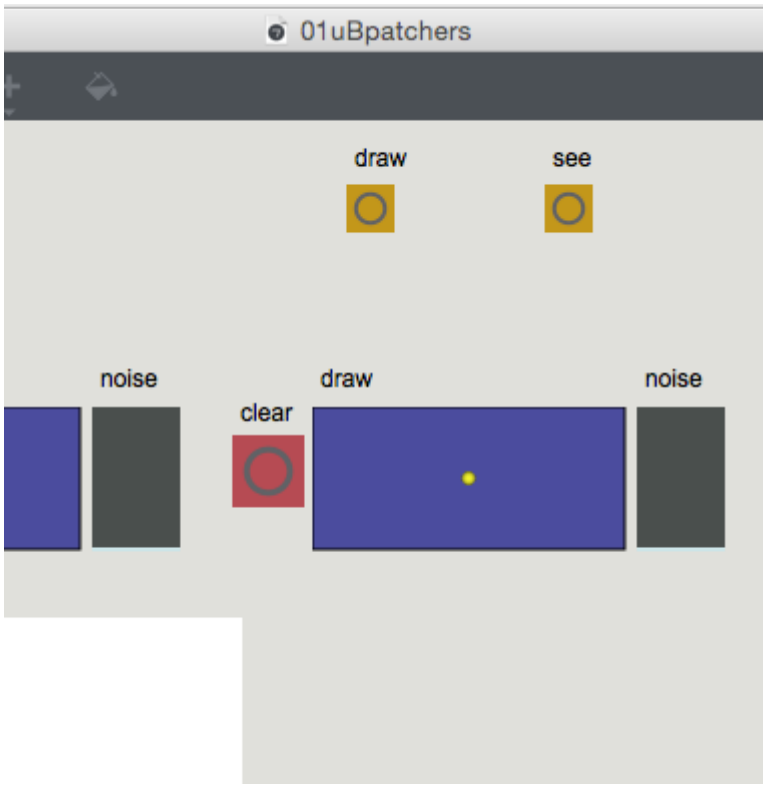
udpreceive

udpsend

Kontrolloppgave til øvelsen *Bpatchers*

Denne øvelsen er en øvelse i å bygge brukervennlige grensesnitt. *Presentation mode* samt objektene *bpatcher* og *thispatcher* gjør det mulig å lage enkle grensesnitt til komplekse patcher.

Gå gjennom øvelsen *Bpatchers* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 31

Lag et grensesnitt ved hjelp av et *bpatcher*-objekt. *bpatcher*-objektet skal ha tre innganger som tar verdier fra tre *number*-objekter utenfor *bpatcher*-objektet, og tre utganger som gir verdier til tre *number*-objekter utenfor *bpatcher*-objektet. Inne i *bpatcher*-objektet skal det være tre slidere og tre *dial*-objekter som tar imot verdier fra inngangene og gir verdier til utgangene. Ved å trykke på en knapp skal man velge om man ser enten sliderene eller *dial*-objektene.

Nye objekter introdusert i denne øvelsen:

thispatcher

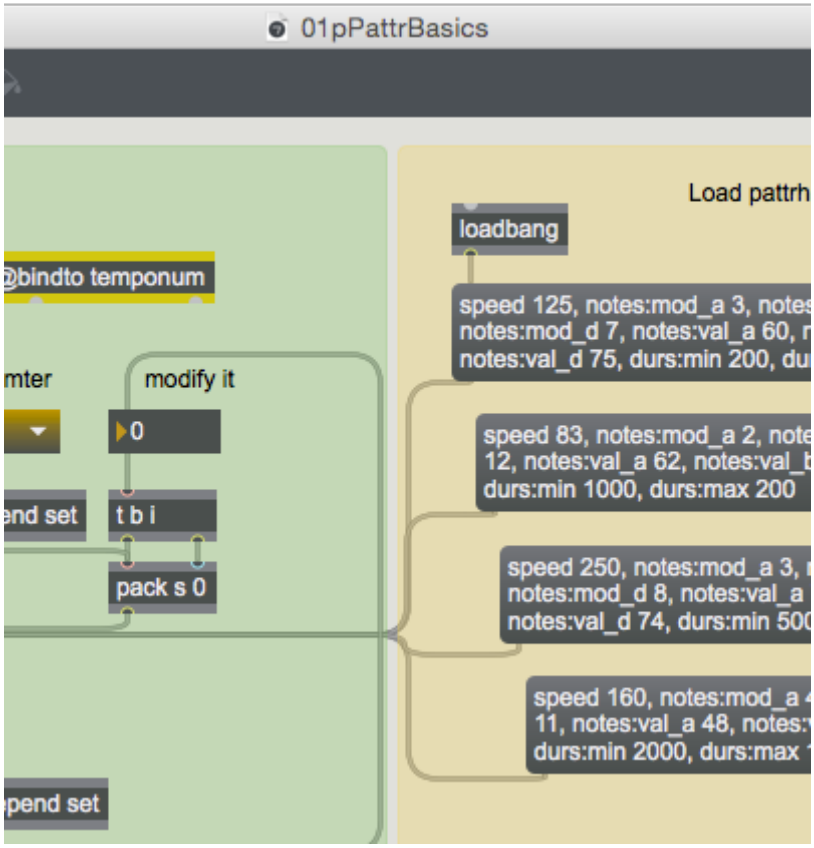
bpatcher

Kontrolloppgave til øvelsen *Pattr Basics*

Øvelsen *Picture UI Elements* gir ulike eksempler på hvordan man kan bruke bildemateriale for å lage egne brukergrensesnitt. Dette er ikke så nødvendig i denne omgangen så vi hopper rett til øvelsen *Pattr Basics*.

Det finnes mange objekter for å styre og lagre data i Max. Noen av de mest brukte er *coll, umenu,* og *preset.* Denne øvelsen går gjennom *pattr*-objektet og ser på hvordan vi kan styre ulike deler av en patch på en mer avansert måte.

Gå gjennom øvelsen *Pattr Basics* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 32

32.1 Lag en patch som spiller av tilfeldige MIDI-noter. Tonene skal komme i en jevn strøm trigget av et *metro*-objekt, men tonehøydene og anslagsstyrkene skal derimot være tilfeldige. Bruk de ulike *pattr*-funksjonene for å sette tonehøyder og anslagsstyrke. Hastigheten til *metro*-objektet skal settes med en såkalt *binding* til *pattr*-objektet. Se på øvelsen en gang til om du trenger mer informasjon om denne funksjonen.

De tilfeldige tonehøydene og anslagsstyrkene skal genereres i hver sin subpatch og styres av *pattr*-objekter.

32.2 Ved å trykke på to knapper skal du kunne velge mellom to ulike innstillinger. Alle verdier er MIDI-verdier fra 0-127

Innstilling 1:

hastighet 80 millisekunder
tonehøyde: min 60
tonehøyde: maks 65
anslagsstyrke: min 100
anslagsstyrke: maks 127

Innstilling 2:
hastighet 250 millisekunder
tonehøyde: min 30
tonehøyde: maks 100
anslagsstyrke: min 0
anslagsstyrke: maks 127

32.3 Til slutt, lag en funksjon som kan lese av den til en hver tid gjeldende hastigheten ved å trykke på en knapp.

Nye objekter introdusert i denne øvelsen:

preset

patcher

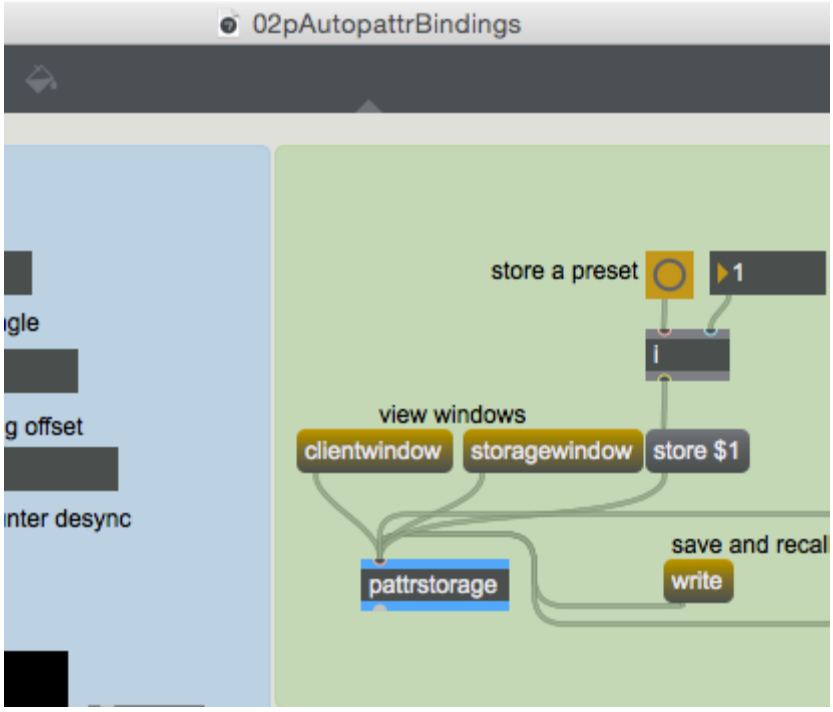
pattrhub

pattr

Kontrolloppgave til øvelsen *Autopattr Bindings*

Denne øvelsen fortsetter der den forrige øvelsen slapp, men går dypere inn i lagring av data med blant annet muligheten for å lagre i XML-formatet og interpolasjon mellom forhåndsinnstillinger (presets).

Gå gjennom øvelsen *Autopattr Bindings* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 33

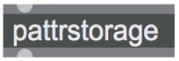
33.1 Ta utgangspunkt i oppgave 36 der du lagde en prosedyre som spilte av tilfeldige MIDI-noter.

Denne gangen skal du ikke benytte objektene *pattr* og *pattrhub* men derimot *autopattr*og *pattrstorage*. *autopattr* og *pattrstorage* er vanskeligere å benytte i forbindelse med subpatcher så vi kvitter oss med subpatchene denne gangen og lager alt i ett patchvindu.

33.2 Vi legger også til muligheten for å lagre og lese forhåndsinnstillinger (presets) som XML-filer samt muligheten for å interpolere mellom forhåndsinnstillinger.

Nye objekter introdusert i denne øvelsen:

pattrstorage



autopattr

BEAP — granulærsyntese med LFO

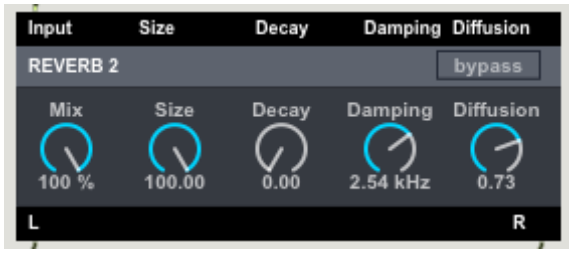
I denne oppgaven skal vi lage en langsomt skiftende dyp klang. For å få til dette skal vi styre granulærsyntesen fra utsiden med en lavfrekvensoscillator (LFO).

La oss fortsette med granulærsyntesepatchen som vi lagde i forrige BEAP-øvelse. Start med å sette *Mix* i *Reverb*-patchen til 0 slik at du hører bedre hva som skjer i granulærsyntesen. Forandre *Duration* til 150 ms og *Random* til 0% i granulærsyntesen slik at vi bruker større korn av lyden om gangen. Sett (position-) *Width* til 1.7 slik at du fokuserer på et kortere utsnitt av lydfilen.

Hent BEAP-patchen *LFO* som ligger under *LFO*-kolonnen. Sett *LFO*-frekvensen (Freq) til 0.01 Hz slik at den beveger seg langsomt. Koble *LFO Sine output* til *Granular Position (+/- 5v)*.
Sett *Position CV* til 100%. *LFO*-patchen styrer nå posisjonen du leser fra i lydfilen.

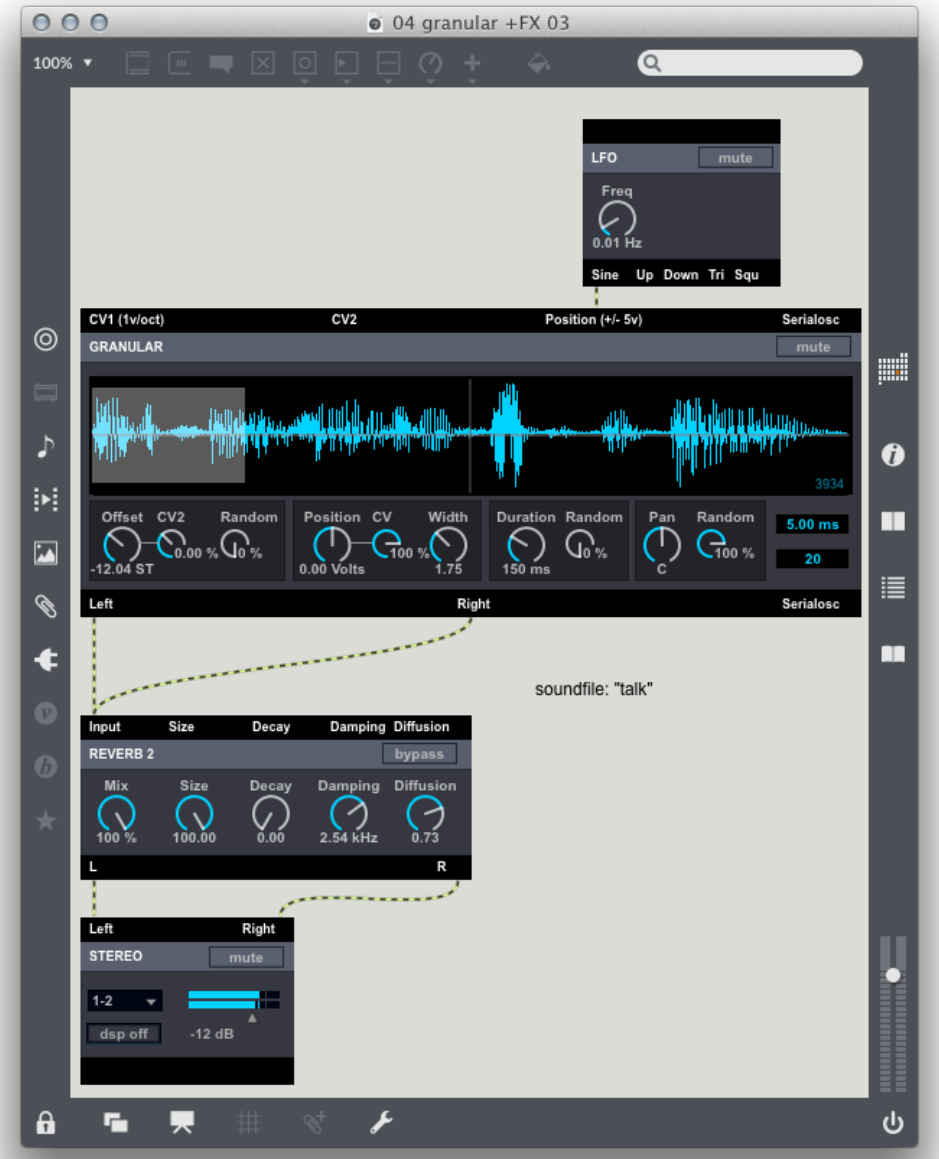
Sett *Grain density* til 5 ms og øk fra 16 til 20 stemmer. Til slutt sett *Offset* til -12 slik at lyden transponeres ned en oktav.

Still inn romklangen slik at vi får et stort rom med lang etterklang med følgende parametre:



Vi setter *Mix* til 100% slik at vi kun lytter til romklangen, *Size* settes til maksimal verdi (100) og *Decay* settes til maksimal etterklang (0). *Damping* settes til 2.54 kHz for å gjøre den mørkere og *Diffusion* settes til 0.73 for å spre stereobildet utover.

Husk at du må velge «Autosave Snapshot» for å lagre innstillingene, og «Embed Snapshot in Parent» for at «Granular»-patchen skal huske hvilken lydfil du har valgt.



Hele patchen