LOLC

Text based collaborative music improvisation system

Sang Won Lee, Jason Freeman, Andrew Colella, Shannon Yao, **Akito Van Troyer**

What’s LOLC?

- LOLC is composed of a set of computer apps and a language.
- Designed to be played collaboratively by laptop musicians.
- Meant to be easy to learn for people with no programming background.
- It also tries to maintain the audience engaged with a built-in visual interface.
- *“Decisions such as the roles ensemble members play, the musical structure of the performance, and the use of a conductor are left to the musicians to decide. “*

01

The language

Simple and concise

The language

01

Simple patterns

Immutable patterns are created from scratch or from other patterns.

02

Schedule based

Commands are simple to understand, short and self descriptive

03

Natural language

Everything is either common music language or as human and concise as possible.

04

Sample based

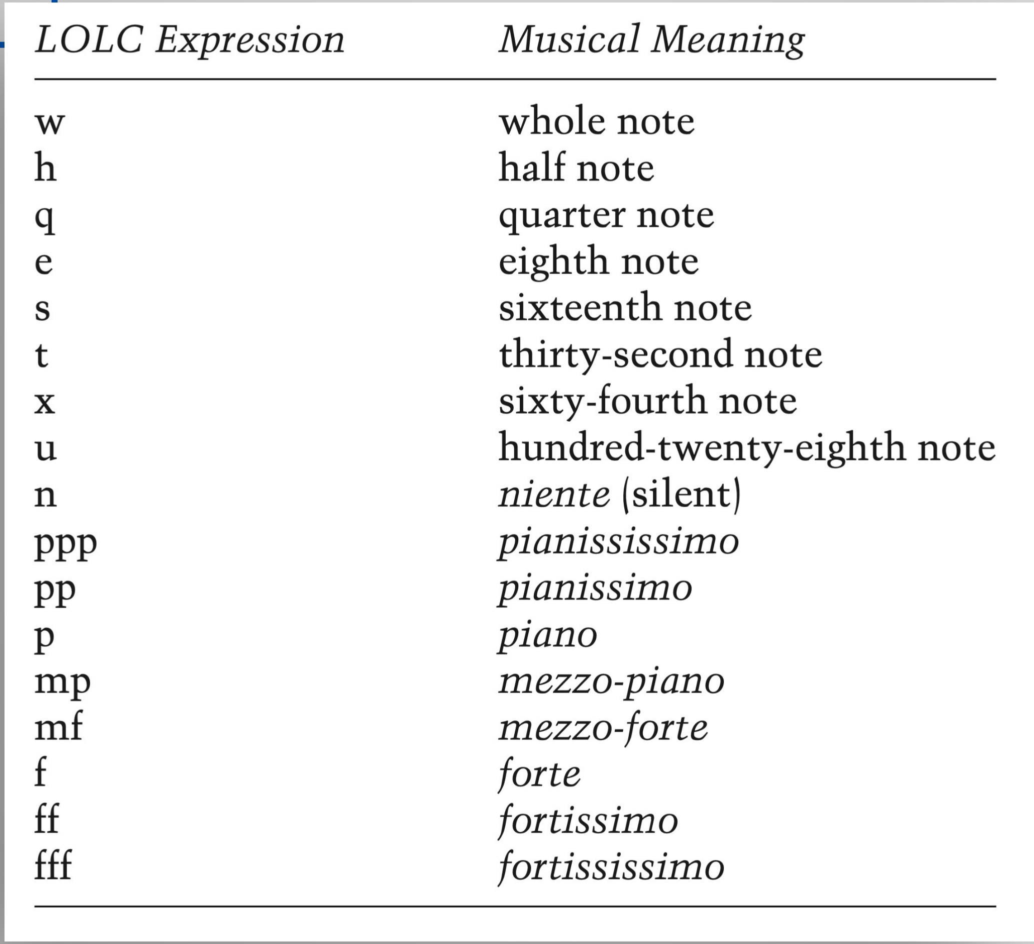
No synthesis is possible in LOLC, everything comes from sound files.

Some code snippets

```
mySound : “sound.aif”


myPattern : mySound[q.ff, q.ff, e.n, e.pp,
e.pp, e.pp]

myNested : myPattern[w,h]
```



LOLC Expression	*Musical Meaning*
w	whole note
h	half note
q	quarter note
e	eighth note
s	sixteenth note
t	thirty-second note
x	sixty-fourth note
u	hundred-twenty-eighth note
n	*niente* (silent)
ppp	*pianississimo*
pp	*pianissimo*
p	*piano*
mp	*mezzo-piano*
mf	*mezzo-forte*
f	*forte*
ff	*fortissimo*
fff	*fortississimo*

Functions

```
myPattern1 : sound1[w,h,h]

myPattern2 : sound2[q,q,q,q]

myConcat : concat(myPattern1, myPattern2)

myTrunc : trunc(myPattern2, 2)
```

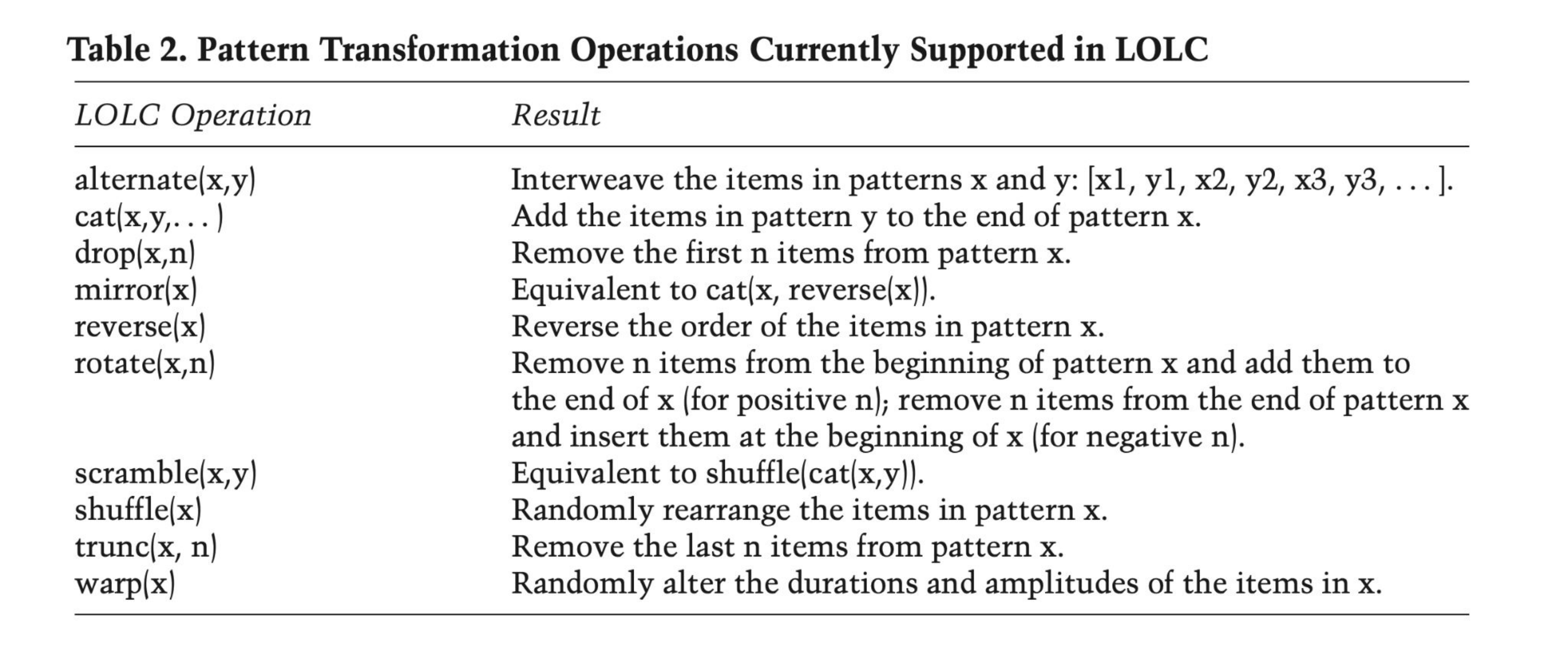


Table 2. Pattern Transformation Operations Currently Supported in LOLC

LOLC Operation	*Result*
alternate(x,y)	Interweave the items in patterns x and y: [x1, y1, x2, y2, x3, y3, . . .].
cat(x,y,. . .)	Add the items in pattern y to the end of pattern x.
drop(x,n)	Remove the first n items from pattern x.
mirror(x)	Equivalent to cat(x, reverse(x)).
reverse(x)	Reverse the order of the items in pattern x.
rotate(x,n)	Remove n items from the beginning of pattern x and add them to the end of x (for positive n); remove n items from the end of pattern x and insert them at the beginning of x (for negative n).
scramble(x,y)	Equivalent to shuffle(cat(x,y)).
shuffle(x)	Randomly rearrange the items in pattern x.
trunc(x, n)	Remove the last n items from pattern x.
warp(x)	Randomly alter the durations and amplitudes of the items in x.

Scheduling

```
play myPattern @ nextBeat

play myPattern @ nextMeasure

play myPattern @ nextHyperMeasure

loop myPattern @ nextMeasure ~16
```

02

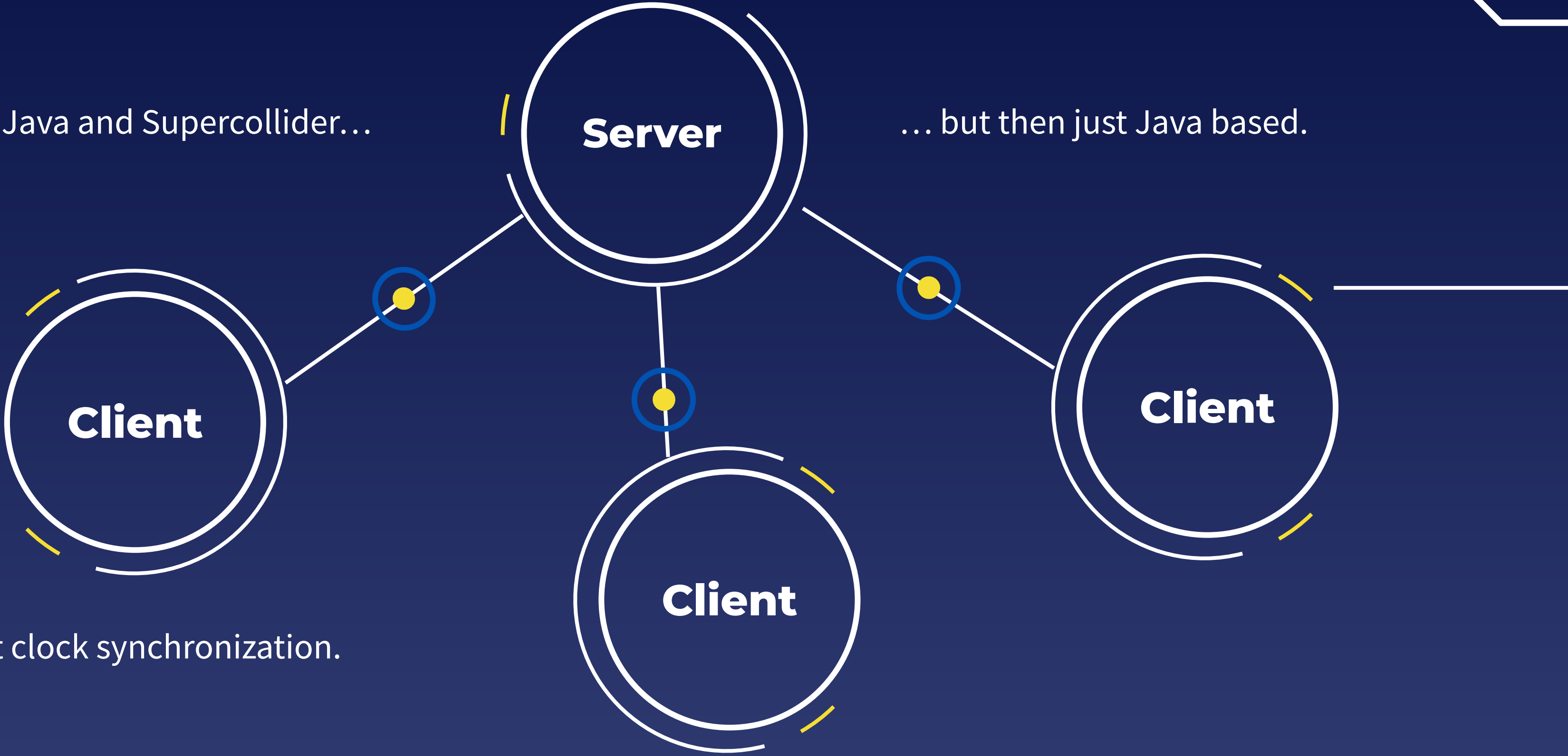
The Implementation

Architecture design

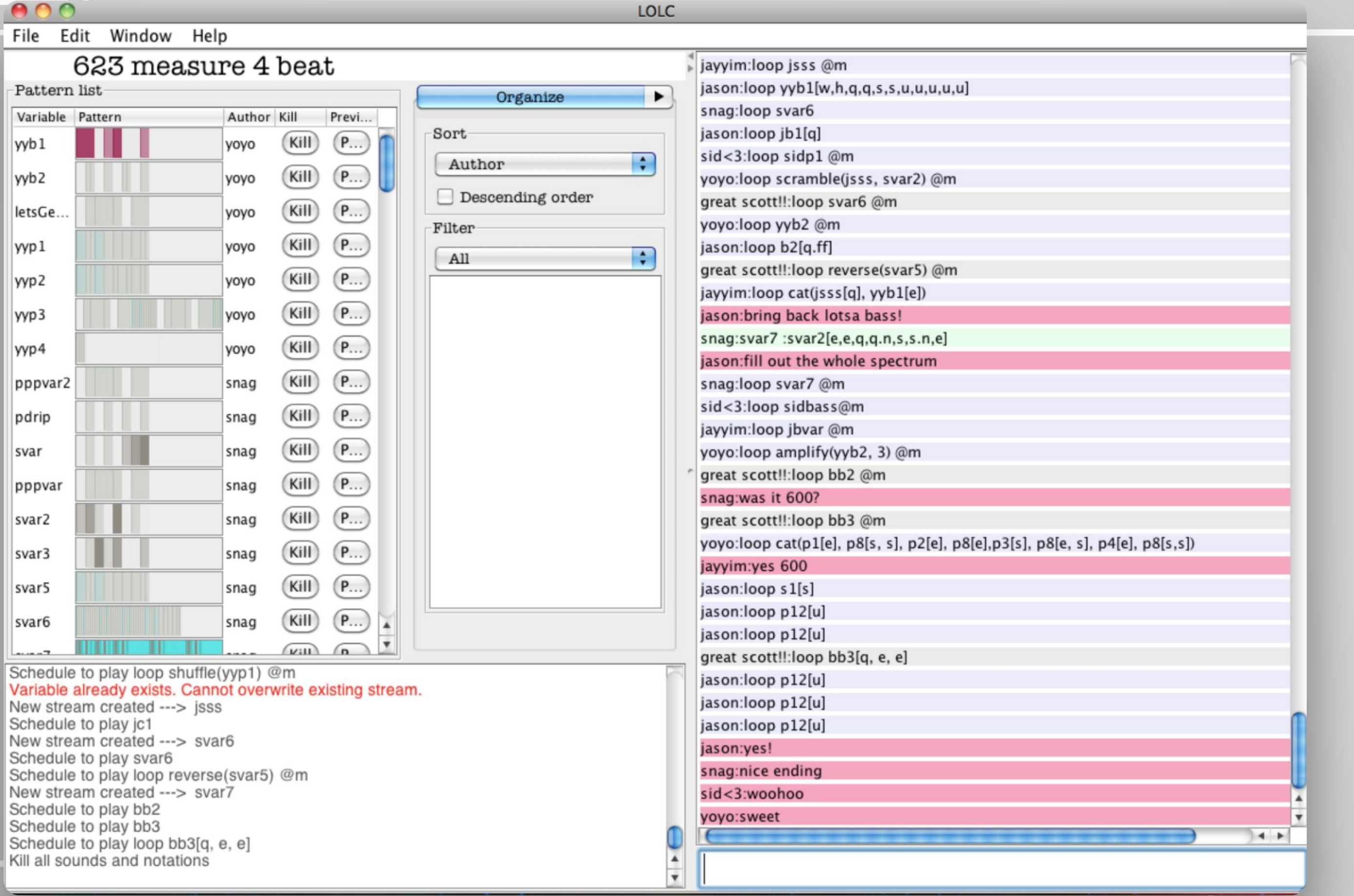
Client-Server Based

Originally in Java and Supercollider…

… but then just Java based.



Internet clock synchronization.



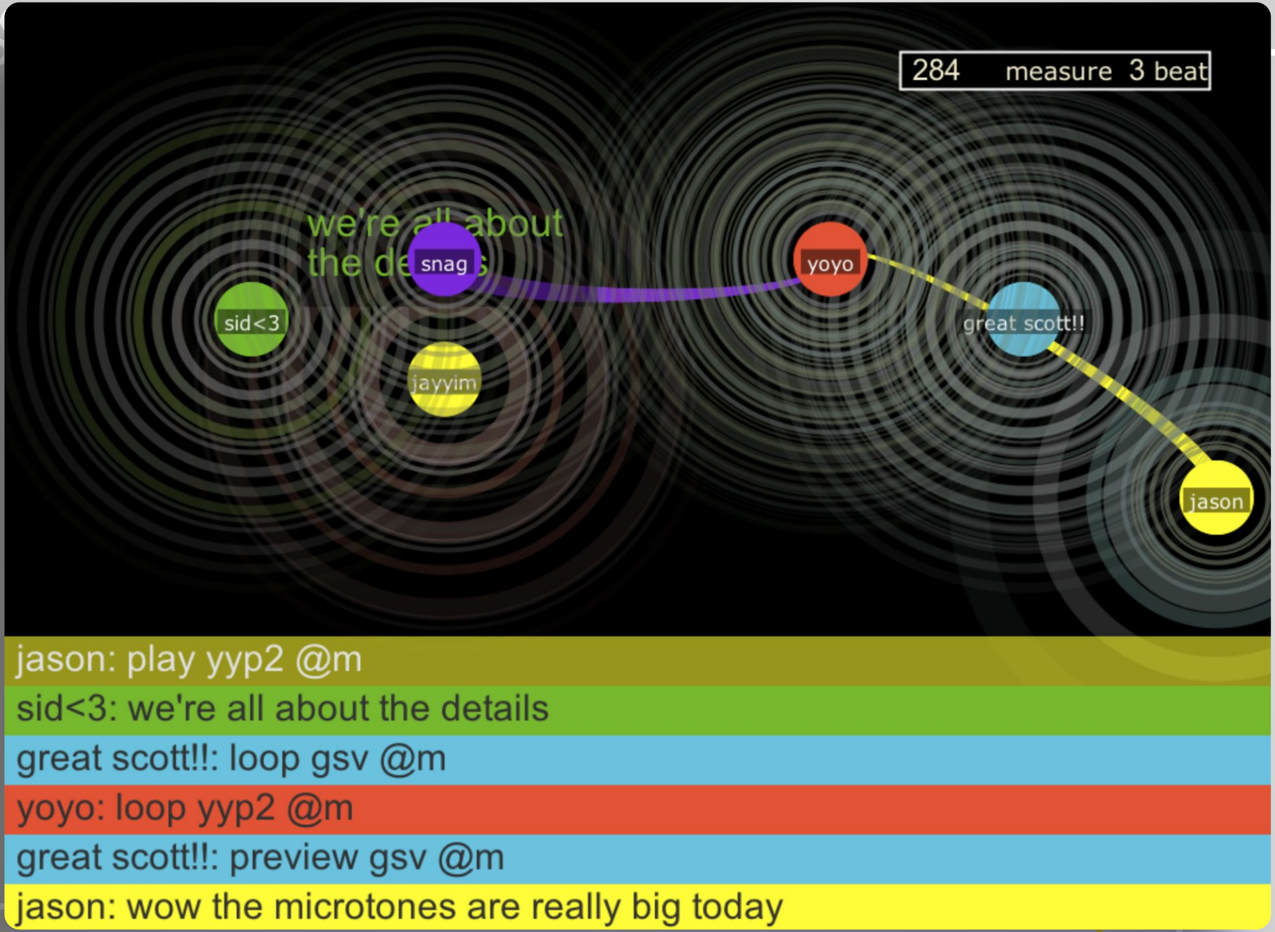
623 measure 4 beat

Pattern list

Variable	Pattern	Author	Kill	Previ...
yyb1		yoyo	Kill	P...
yyb2		yoyo	Kill	P...
letsGe...		yoyo	Kill	P...
yyp1		yoyo	Kill	P...
yyp2		yoyo	Kill	P...
yyp3		yoyo	Kill	P...
yyp4		yoyo	Kill	P...
pppvar2		snag	Kill	P...
pdrip		snag	Kill	P...
svar		snag	Kill	P...
pppvar		snag	Kill	P...
svar2		snag	Kill	P...
svar3		snag	Kill	P...
svar5		snag	Kill	P...
svar6		snag	Kill	P...

```
Schedule to play loop shuffle(yyp1) @m
Variable already exists. Cannot overwrite existing stream.
New stream created ---> jsss
Schedule to play jc1
New stream created ---> svar6
Schedule to play svar6
Schedule to play loop reverse(svar5) @m
New stream created ---> svar7
Schedule to play bb2
Schedule to play bb3
Schedule to play loop bb3[q, e, e]
Kill all sounds and notations
```

```
jayyim:loop jsss @m
jason:loop yyb1[w,h,q,q,s,s,u,u,u,u,u]
snag:loop svar6
jason:loop jb1[q]
sid<3:loop sidp1 @m
yoyo:loop scramble(jsss, svar2) @m
great scott!!:loop svar6 @m
yoyo:loop yyb2 @m
jason:loop b2[q.ff]
great scott!!:loop reverse(svar5) @m
jayyim:loop cat(jsss[q], yyb1[e])
jason:bring back lotsa bass!
snag:svar7 :svar2[e,e,q,q.n,s,s.n,e]
jason:fill out the whole spectrum
snag:loop svar7 @m
sid<3:loop sidbass@m
jayyim:loop jbvar @m
yoyo:loop amplify(yyb2, 3) @m
great scott!!:loop bb2 @m
snag:was it 600?
great scott!!:loop bb3 @m
yoyo:loop cat(p1[e], p8[s, s], p2[e], p8[e],p3[s], p8[e, s], p4[e], p8[s,s])
jayyim:yes 600
jason:loop s1[s]
jason:loop p12[u]
jason:loop p12[u]
great scott!!:loop bb3[q, e, e]
jason:loop p12[u]
jason:loop p12[u]
jason:loop p12[u]
jason:yes!
snag:nice ending
sid<3:woohoo
yoyo:sweet
```



jason: play yyp2 @m

sid<3: we're all about the details

great scott!!: loop gsv @m

yoyo: loop yyp2 @m

great scott!!: preview gsv @m

jason: wow the microtones are really big today

03

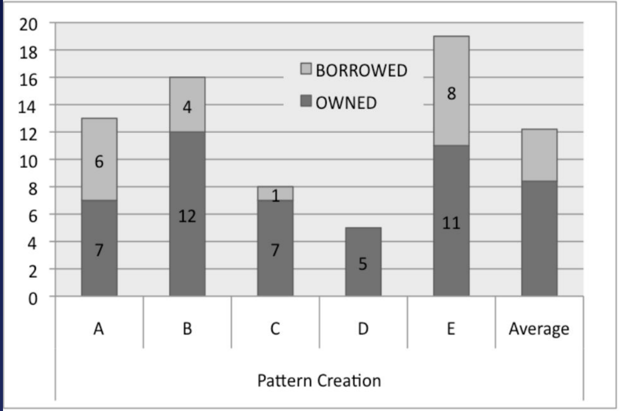
The Performance

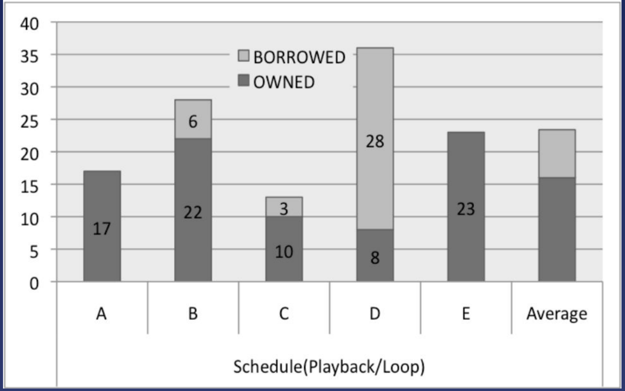
How people tend to use LOLC

How did the musicians react?

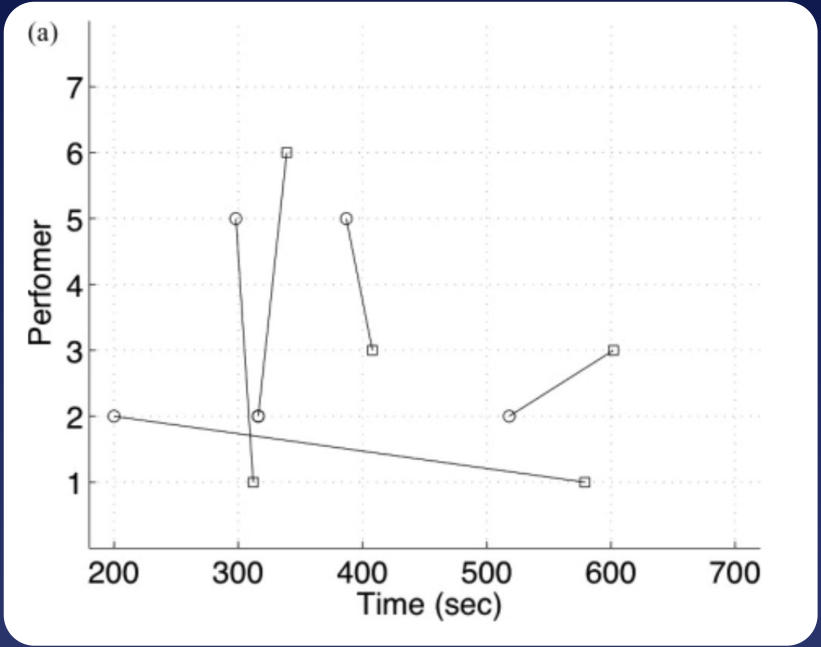
After giving some classically trained musicians a month to prepare, and 12 hours of rehearsal, these were the results at the performance:

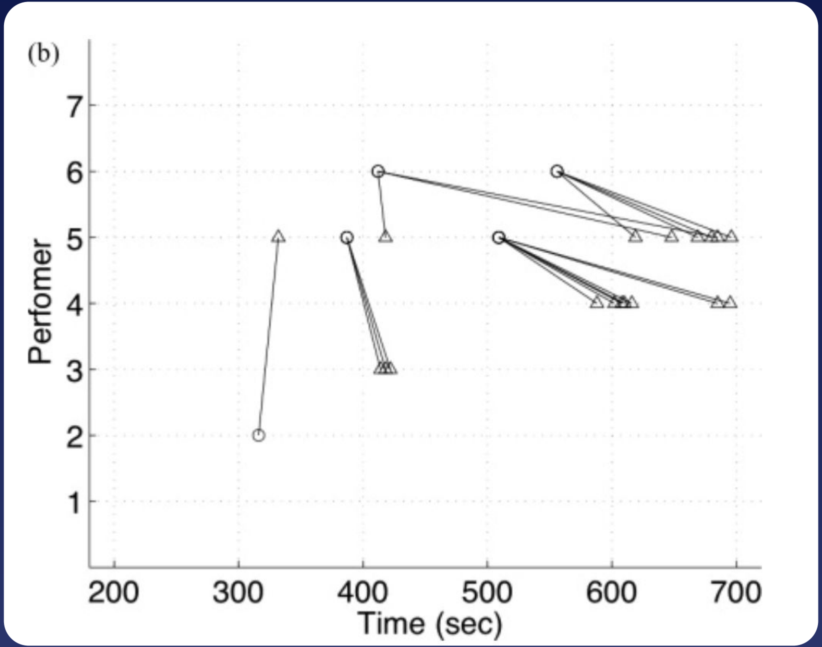
- Musicians successfully got the grasp of the system with relative ease.
- Musically interesting results.
- Mostly stuck in a loopy way of composition.
- Audience didn't get nearly anything of the code, but enjoyed the chat.





How did the musicians react?





04

Final thoughts

Design paradox and future work

“There is, perhaps, an inherent design paradox here. Performing musicians tend to prefer a concise syntax that requires minimal typing, whereas audiences tend to have difficulty understanding text that is not sufficiently verbose”

Future work

Non-programmers

Keep researching on how non-programmers learn and interact with LOLC, exploring its use in education.

Sight reading

Explore the possibility of implementing a version of LOLC where each laptop is paired with a sight reading musician, and the patterns are displayed as classical notation.

Thank you