JSoLangs: ephemeral esolangs in a collaborative live coding environment
For CSDH-SCHN 2021: Making the Network

David Ogborn, McMaster University
Clarissa Littler, Portland Community College
Kate Sicchio, Virginia Commonwealth University

This paper documents and reflect upon the initial stages of an experiment in creating ephemeral live coding languages — "JSoLangs" — for diverse artistic, educational, and critical purposes. The experiment takes place in the context of the larger Estuary project: a browser-based, collaborative platform that allows multiple, distinct live coding notations/languages to interoperate.[1] Estuary provides a curated set of base live coding notations that target distinct audiovisual results. For example, the Punctual language, available within the Estuary environment provides an economical notation for simultaneously creating synthesized sound and visuals,[2] while the CineCer0 language provides a declarative syntax for controlling the playback of videos and kinetic typography within the web browser. Many other live-coding focused languages are available within Estuary, which aims to provide a space where "multi-lingual" live coding can be explored without needing to install software (other than a suitable web browser). Estuary can be accessed at https://estuary.mcmaster.ca.

An emerging feature of the Estuary platform is the ability to create ephemeral "esolangs" [3] on the fly, within a collaborative Estuary ensemble. These "JSoLangs" (pronounced jay-es-so-langs, to rhyme with "esolangs") take the form of small (or not so small) JavaScript programs that transpile live coded text into one or more of Estuary's underlying languages. Esolangs (short for "esoteric programming languages") are (generally) small programming languages designed for experimental purposes, often with the intent to make programming difficult. The context of live coding (wherein artist-programmers produce aesthetic results by making and changing small programs on the fly, often with an audience with whom both code and its result is shared) is a unique context for creating and exploring the possibility of such esolangs. The live coding artist programmer who makes an esolang may create and modify such a language in order to change the interface presented to themselves (or their collaborators), or they may create and modify such a language in order to communicate differently to an audience. It seems likely that in most cases the "difficulty" factor that is commonly associated with esolangs will be less emphatic in the case of live coding esolangs/JSoLangs (indeed, in all three of the examples discussed below the languages arguably make specific aesthetic results *easier* to achieve). In Estuary's implementation, JSoLang's take the form of grammars/parsers using the peg.js library [4], a widely used JavaScript parsing library. Successful parsing is expected to produce a valid text program in some language that is comprehensible to Estuary.

During the first half of 2021, the authors of this paper have created a series of JSoLangs in the Estuary platform, in order to reflect on the significance of such ephemeral languages to distinct purposes, such as artistic expression, learning to code, and the subversion of norms both in terms of mainstream discussions of general purpose programming languages as well as in terms of the somewhat more rarefied discourse and practices surrounding live and creative coding. The remainder of this paper offers brief descriptions of three specific JSoLangs we've created, and at the CSDH-SCHN 2021 conference we'll do a short live collaborative performance using all three JSoLangs together.

Rando (created by Clarissa Littler)

The simple JSoLang I made was inspired by LISP-like macro replacement, and is built on top of TidalCycles [5] — a language for creating patterns of sound — but the JSoLang framework is used to add a new operator: `?`

The `?` operator, at parse time, is interpreted as a randomly chosen sample from a predefined list of samples. This means that whenever you use `?` you don't know exactly what sample you're going to get, but you have at least some idea. I like using this as a way of focusing on the structure for parts of the code rather than spending time thinking about the samples themselves. I can still pick samples the way I normally would in my workflow, but the language is simple enough that I can modify it in the middle of a performance to change the samples or how they work.

Demo video: https://www.youtube.com/watch?v=cHp-x8ALD4M

dr0nezer0 (created by David Ogborn)

dr0nezer0 is a JSoLang created by David Ogborn, over the underlying Punctual language (also created by David Ogborn). Punctual is a Haskell-ish live coding language in which signal processing graphs are described with an economical notation. Two key features distinguish Punctual: (1) notated graphs can be translated into both audible and visual results (Web Audio API synthesis effects, and GLSL fragment shaders), and (2) Punctual provides notations for describing the transitions between the old version of a program and a new version of the same program (a concern highly distinctive of live coding as a practice). dr0nezer0 arose from the desire to further emphasize Punctual's ability to target both audible and visual results, by creating an esolang in which ultra-economical expressions produce both sound and visuals: pitched continuous chords (drones) accompanied by circular imagery that is produced and moves in relation to the pitch of the chords. Two additional criteria for the new JSoLang was that it would include some recursive syntax (eg. arithmetic expressions) and that it would use numbers and symbols, no letters and words (a criterion that may be relaxed as it evolves).

Here are some examples of dr0nezer0 programs, as the JSoLang is currently defined (each line is intended to be a separate example, and text after -- is a comment):

```
[60,64,67] -- a chord
[60,64,67] + 3 -- the same chord transposed up three semitones
[60,64,67] + [0,0.25] -- the same chord transposed by 0 and by 0.25 semitones
```

Demo video: https://youtu.be/4g6gSC_M0Ww
GitHub repository: https://github.com/dktr0/dr0nezer0

Studio//Stage (created by Kate Sicchio)

Studio//Stage is a JSoLang by Kate Sicchio that sits on top of the CineCer0 language in Estuary. The aim is to use dance specific vocabulary to create a programming language for the creation of screen dance compositions (dance_cinecer0_sidexside.mov). The idea of Studio//Stage is that movement in the form of video clips is developed and crafted through the development of the JSoLang in the "studio" side of the live coding, mapping dance terminology to CineCer0. The final video performance

that is viewed is coded in the “stage” side of language. Both the studio and the stage can be manipulated in performance to create a video dance that explores timings, loops, phrases and locations on the screen, allowing for algorithmic choreography to emerge in real-time and the development of the “programmer as choreographer” [6].

The networked capabilities of Estuary allow for an expansion of the choreographic process into a real-time collaborative method. One choreographer can be shaping the movement material in the “studio” and the another choreographer can be composing the piece in time and space on the “stage”. This type of improvisational structure is rare in algorithmic choreography as it is normally programmed by a solo coder. This form of collaboration will be explored further in future work with Studio//Stage.

Demo video: https://youtu.be/UQ1l8PRkcM4
GitHub repository: https://github.com/sicchio/studio-stage

References:

[1] David Ogborn, Jamie Beverley, Luis Navarro del Angel, Eldad Tsabary, Alex McLean, Esteban Betancur (2017). “Estuary: Browser-based Collaborative Projectional Live Coding of Musical Patterns.” Proceedings of International Conference on Live Coding, Morelia, Mexico.

[2] https://github.com/dktr0/Punctual.git

[3] https://esolangs.org

[4] https://pegjs.org/

[5] McLean, Alex. (2014). “Making Programming Languages to Dance to: Live Coding with Tidal.” Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design. FARM '14.

[6] Sicchio, Kate. (2010). “Exploring the software programmer as choreographer.” Digital Resources for Humanities and Arts, September 2010, London.