Estuary 0.3: Collaborative audio-visual live coding with a multilingual browser-based platform

David Ogborn
McMaster University
ogbornd@mcmaster.ca

Jamie Beverley
jamie_beverley@hotmail.com

Nicholas Brown-Hernandez
McMaster University
brownhen@mcmaster.ca

Alejandro Franco Briones
McMaster University
francoba@mcmaster.ca

Bernard Gray
bernie@grbt.com.au

Alex MacLean
maclean199@gmail.com

Luis N. del Angel
McMaster University
navarrol@mcmaster.ca

Kofi Oduro
illestpreacha@outlook.com

Spencer Park
spinnr95@gmail.com

Antonio Roberts
hellocatfood@gmail.com

Jessica Rodríguez
McMaster University
rodrij28@mcmaster.ca

Kate Sicchio
Virginia Commonwealth University
ksicchio@vcu.edu

D. Andrew Stewart
University of Lethbridge
contact@dandrewstewart.ca

Carl Testa
carl@carltesta.net

Eldad Tsabary
Concordia University
eldad.tsabary@concordia.ca

ABSTRACT

Estuary is a browser-based platform for live coding that allows a heterogeneous collection of live coding languages to be used together in collaborative "ensembles". This paper begins with a broad outline of the history of Estuary's development, including discussion of the philosophies and accountabilities that guide that development. We then present two of the main directions in which work on Estuary has been concentrated in recent years: (1) towards supporting audiovisual live coding (and not merely musical live coding) through the development and inclusion of languages focused on visual results (in close connection with musical concepts), and (2) towards supporting diverse use cases by evolving into a modular "sandbox" where live coding languages, widgets (including some focused on intra-ensemble communication), and media resources are brought together "on-the-fly" in flexible ways. Reflections on specific applications of Estuary in different contexts are interspersed throughout, with a penultimate section focusing on some further applications, largely in educational settings. The paper concludes with brief remarks about directions for future work.

Web Audio Conference WAC-2022, July 6–8, 2022, Cannes, France.

1. INTRODUCTION

Estuary is a browser-based platform for collaborative live coding that has been developed and in active use since 2015. Estuary's affordances are characterized by an emphasis on mixing heterogenous live coding languages (as well as other interfaces to media computing) in collaborative, networked "ensembles". Estuary's ensembles can be used by completely co-located groups (as a way of facilitating collaboration and sharing access to hardware), by completely distributed groups (as a way of working together despite geographic and logistical constraints), and by hybrid groups that mix these two possibilities (such as a co-located group with a few members "dialing in" remotely).

The earliest work on Estuary [1] began with the goal of exploring structure and projectional editing around the TidalCycles live coding language [2], producing a series of different structure editors characterized by minimal keyboard usage, the use of blank space as an interface, and a mixture of notations at different levels of programming liveness [3, 4]. From that starting point, work on Estuary then returned to an abiding interest in multiple, heterogenous, text languages, typed with a keyboard, as a key interface for engaging with the possibilities of live coding. Estuary's early evolution into an environment for "multilingual" live coding was inspired by the recognition that facilitating an individualized choice of programming interface, within a larger collaborative setting, would allow people to choose languages that best suit their purpose or situation, without thereby having to forego collaborating with others making different choices. A member of a group might choose to work with a particular language because of an existing level of comfort or familiarity with it, or they might choose a language on account of its close fit to a particular result that is required in a certain moment (eg. choosing a language that facilitates making drone-like continuous sonic textures, when such textures are desired), or – since live coding is often not only about the result but also about the way the code is shared with an audience

– they might choose a language precisely for what it "looks like" and what it denotes and/or connotes for an audience. At another level, a multilingual interface producing multilingual performances (and other forms) of live coding, may help to unsettle all-too-definitive conceptions of what programming and languages are [5].

The live coding languages available within Estuary have changed over the years, as languages have been added and removed in response to specific artistic projects, directions of research, and the evolution of relationships around the software. At the time of writing, six different "base" text-based live coding languages are present within the Estuary platform (MiniTidal, Punctual, CineCer0, TimeNot, Seis8s, Hydra). However, this number is not that definitive of the extent of programming language and interface choices provided by Estuary, as the platform provides for a growing number of forms of modular extension, including but not limited to live-coded JSoLangs (see below and [6]), that are user-defined languages transpiled into other supported languages. Estuary also provides a View system that allows for very flexible configuration of the way different programming interfaces are combined, on-screen, for a particular artist programmer or viewer, and a system whereby new (or different versions of) base languages are added by pointing directly to web-located JavaScript libraries (that are "readily available for", but not inherently "built in to", Estuary) is rapidly emerging (see below, Future Work).

Estuary consists of a browser-based client that communicates with a server, with almost all of the ongoing communication between client and server being for the purpose of sharing collaborative user actions between different clients in the same "ensemble" (ensemble being the Estuary-specific vocabulary for an environment where one or more users are able to change live coding programs and other interfaces, with both their actions and the results of those actions being shared to all of the other clients connected to the same ensemble). All of the translation of live coding programs into sonic and visual "results" is done by the client, working in the context of the web browser and the APIs it provides (such as the Web Audio API and WebGL). The server is responsible only for distributing the minimal information required to keep programs and other interfaces "in sync" (eg. displaying and rendering the same text program).

Estuary's basic model for editing, evaluating, and executing shared code aims to produce identical (or nearly identical) results as rendered by different clients/machines. Several aspects of the model (and the broader ecosystem) contribute to this relative determinism of the output: (1) The system time at which updated code (i.e. a "new program") is evaluated is treated as a significant aspect of that program, which enables specific languages to use that information to align results; (2) many of the languages available through Estuary generate results in deterministic ways given a "timeline" defined by a prevailing metric grid and the passage of system time (for example, Punctual's oscillators are aligned in specific ways against that timeline depending on the timestamp of evaluation time, and pseudo-random number generation in TidalCycles and Punctual is a deterministic function of position within the timeline); and (3) the evaluation of code is typically subject to various forms of quantization, so that changes in the rendered results are, intentionally, deferred into the near future. Particular clients/machines only need to receive new code (evaluated by others, elsewhere) before the local, system time at which that new code would produce different results. This leads to an interesting situation in which rendered results at different locations appear to "escape" the effects of network latency, while latency still (unavoidably) exists in terms of the communicative interactions between the humans in the system. Synchronization in Estuary is tied to the operating system's clock on each machine. In the case of distributed ensembles, this works very straightforwardly – because the results are rendered independently at each location, minor drifts in the clock synchronization will likely be imperceptible and/or irrelevant. In the co-located case, the system time is not usually synchronized enough to allow, for example, the audio output from two co-located machines to "line up" in a given perceptual frame of reference. This is not usually a limitation, however, because it is common in co-located uses of Estuary for one client/machine to be designated as the "rendering machine", with all other co-located users/clients disactivating their audio output completely.

Both the client and the server are developed in the Haskell strict functional programming language, using the GHCJS compiler and the reflex-dom library for the browser-based client, and the GHC compiler for the server. This approach allows code to be reused across the server and client, and supports integration, on an on-going basis, with Haskell-based live coding projects like TidalCycles. Additionally, writing parsers in the style of "combinatorial parsing" [7] is a pleasure in Haskell, and so our Haskell-based ecosystem seems well-suited to encouraging play and experimentation with the design of live coding languages. Two libraries over the Web Audio API also play a key role: MusicW [8], which provides a higher-level interface over the Web Audio API ; and WebDirt [9], which provides a simple interface for parameterized sample playback (originally based on a not-very-faithful re-implementation of TidalCycles' sampling engine's Dirt and SuperDirt). A third library, haskellish [10], provides a high-level interface for parsing languages that are, in some sense, "like" Haskell, and is used by various configuration notations within Estuary as well as by quite a few of the live coding languages available within Estuary. The Estuary source code, as well as that of all the libraries it uses, is licensed on an open-source basis.

At the time of writing, an up-to-date deployment of Estuary is maintained continuously at https://estuary.mcmaster.ca for use by a global audience of live coding artist programmers and learners. Estuary is produced and maintained using specific (ostensibly "public") resources, by specific individuals and through specific relationships, and in connection with specific territories (including but not limited to the territories at the south-west end of Lake Ontario, where McMaster University is situated, where the most direct work on Estuary's code base happens, and where the aforementioned server is located). This situation gives rise to responsibilities and accountabilities that exceed the simple act of sharing the software. As currently implemented, Estuary more or less depends on access to "very good" computers, as the browser environment is not the most performant place to render computer music and generative visuals. This in turn implies a kind of complicity with, or dependency on, "cutting edge", "disposable" computing with a further connection to careless or destructive resource extraction – like any live coding platform Estuary participates in a "full stack" that runs from mining and minerals up to interfaces and user effects [11]. As such, ongoing work on the challenge of making the elements of the platform perform more efficiently is a key priority: a more efficient platform can be used by more people, and will contribute less to an unnecessary and harmful economy of "planned obsolescence". A goal as the project moves forward: Estuary should work for today's computers, for an increasing number of "yesteryear's" computers, and should not require "the computers of tomorrow".

Estuary has also emerged at a specific point in the evolution of the Internet, where the majority of websites visited by users participate in various forms of user-specific surveillance. Estuary responds, in this context, with a security model that is based on collectively shared passwords rather than individually identified "users". The aim of this model is to leave it to people, independently, to manage who they can trust enough to share a collaborative space with (and how), rather than for the platform to impose a model of who can be trusted attached to identified users. As Wendy Hui Kyong Chun has elaborated, the security of Web 2.0 with its database of users, is no security at all [12]. It remains to be seen how Estuary's "user-less" security model will evolve with the platform, but the intention is to resist, as much as possible, the use of individual user identification practices to solve trust and safety problems.

2. FROM MUSICAL TO AUDIOVISUAL LIVE CODING

Estuary development over the past five years has been characterized by two principal directions: (1) towards supporting audiovisual live coding (and not merely musical live coding) through the development and inclusion of languages focused on visual results, in close connection with musical concepts; and (2) towards supporting diverse use cases by evolving into a modular "sandbox" where live coding languages, widgets (including some focused on intra-ensemble communication), and media resources are brought together "on-the-fly" in flexible ways. This section describes Estuary's growing support for visual, audiovisual, videomusique (etc) results by introducing three of the current slate of "base" languages that are strongly oriented towards visual results: Punctual, CineCer0, and Hydra. There is no particular instrumental rationale that has driven the choice of these three languages rather than others. Most simply, these languages have been "incorporated" into Estuary, during this recent period of work, because of the specific interests and questions of the people working on Estuary at the moment (in the future, as relationships around the language evolve, the selection of languages will presumably evolve as well; see also the comments below in section 5, Future Work, re: including languages as externally defined exo-langs). Punctual and Hydra share a common orientation towards 2D video synthesis, but with quite different notational paradigms. CineCer0 is focused on the declarative "presentation" of pre-existing video files and formatted text.

2.1 Punctual

The first visual language to be added to Estuary (after a series of short-lived experiments with small visual languages based on HTML canvas operations, anyway) was Punctual, created and maintained by David Ogborn. Punctual will be given a more substantial research account elsewhere. However, it is useful to give a brief account of Punctual in this article both since it has been a visible part of many audiovisual live coding performances using Estuary, and because some of its features speak to artistic possibilities that are specific to collaborative audio-visual live coding (above all: simultaneous generative visuals by multiple programmers, with positional differentiation or transparency), and which will no doubt be unpacked further by various languages, ensembles, and platforms.

Punctual is a web-based language for audiovisual live coding, available both within the Estuary platform, and separate from it as (non-collaborative) web-based standalone [13]. One of Punctual's defining features is that it explores the possibilities for a specifically audiovisual form of creative activity, by proposing unified notations that are simultaneously translated into both visible and audible results, realized via WebGL and the Web Audio API, respectively. Another equally important defining feature of Punctual is that it proposes notations that facilitate the specification and combination of multiple channels of information (in both the sonic and visual domain), both in the form of output layers that facilitate large numbers of channels of information being treated as results and in the form of operations that interpret multiple channels of information as spaces of possibility to be explored combinatorially. Punctual also grows out of practice with SuperCollider's JITLib library [14], which has inspired Punctual's notations for controlling how successive versions of a program succeed one another, and also from the example of live coding environment The Force (for generative visuals) [15], which like Punctual involves creating a fragment shader that draws on a flat surface with connections to analysis of audio input/output.

In the following example of an audiovisual Punctual program (code displayed together with visual result), a set of 8 low frequency oscillators (labelled x1, y1, x2, and y2) are used to influence both the motion of thin, differently coloured lines, and micro adjustments in the tuning of a dense, chordal drone:

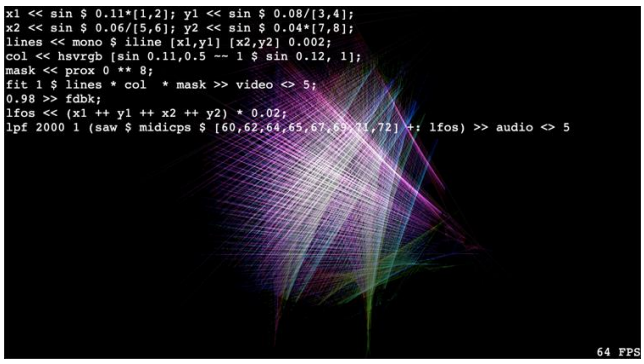

Figure 1. A Punctual program creating audio and animation.

2.2 CineCer0

The next visual language to be added to Estuary was CineCer0, developed by the Estuary development team. The initial impulse for the new language was to have a way of live coding the playback of pre-existing video files, with the ability to control the geometric positioning/scaling of the videos, as well as the ability to align the timing of the videos in various ways. The CineVivo project [16], which allows for the live coding of videos (with a procedural base syntax, over which various forms of token substitution/macros are often used in practice) was an important point of reference at the beginning of work on CineCer0 (a fact memorialized in the similarity of the names). Currently, CineCer0 supports the playback and manipulation of video and image files (including but not limited to SVG files), as well as the display of stylized text information, within Estuary. It appears to be particularly well-suited to use in settings where narrative and storytelling are a priority, although it has been applied in more "abstract" or improvisational settings as well.

A key early design decision was to make the syntax of the new language (CineCer0) as declarative as possible. There were multiple motivations for this emphasis on declarative syntax: (1) our experience with highly declarative live coding environments like TidalCycles, Punctual (see above), or arguably SuperCollider's pattern library/objects, suggests that declarative programming is highly suited to artistic settings, particularly fast-moving improvisational ones; and (2) the relative simplicity with which

declarative semantics can be synchronized/unified over distributed collaborative ensembles. Simply put, declarative semantics make it easier to design a system where someone can join once things are already in process (or rejoin, for example, if they encounter computing or network problems) and be "guaranteed" to experience the "same" things as at other locations, because sharing the "declaration" is enough to produce identical results – one does not have to share both a procedure (and its history) and also, somehow, fast-forward to some point in that procedural history.

Figure 2. From Estuary's *CineCer0* tutorial

The core mechanism of CineCer0's implementation is simply to add video and image objects, or text "divs", to the browser's document object model, and then to modulate their properties on the fly. CineCer0 appears to be highly performant with images and text, but decoding simultaneous video files can be challenging depending on the file and available computational resources. An effective workaround strategy for this difficulty is to produce and use lower resolution video files, and this strategy also tends to work well with the settings and applications in which CineCer0 tends to be used – often artist programmers will use CineCer0's emphasis on sizing and positioning of videos to present them in only specific parts of the screen at a time, thus leaving space for code and other visual results to be visible, and thus not "requiring" the full resolution videos

2.3 Hydra in Estuary

Hydra is a browser-based live coding environment [17], created by Olivia Jack, that employs JavaScript to compose networks of video synthesizers, with a strong orientation towards the results space of historical analog video synthesis. An energetic global community has sprung up around the language, freely sharing short programs that create striking generative visual results. Motivated by a desire to engage with that community, we decided to add support for Hydra to Estuary. However, the unique context of the Estuary environment complicates what might at first glance seem to be a relatively simple matter... Hydra is a bunch of JavaScript modules, that run in the web browser, so can't we just "add" them and stir?

Rather than just use JavaScript's built-in evaluation features to run Hydra natively, we have implemented a system where possible Hydra "programs" are modelled with an abstract syntax tree (AST) that is then translated (as an effect of Estuary's rendering engine) into calls to the "external" Hydra code. This much more indirect way of using Hydra's features is essentially a way of protecting collaborative ensembles against the effects of arbitrary JavaScript execution. For example, if, in the context of writing a Hydra program, in a collaborative ensemble, someone were to insert a never-ending computation, that computation would, effectively, halt the performance for everyone connected to that ensemble. Reloading the ensemble would not help – the ensemble would still contain the "problematic" code. Moreover, it is highly impracticable (possibly: impossible) to identify such problematic code analytically and prevent its execution (cf. the halting problem [18]).

By having a strict (if more limited) model for what the possibilities of Hydra are, we are able to avoid those that might be computationally intractable. Unfortunately, the cost of this in the short-term is that our model does not include some possibilities that are commonly employed by Hydra users – particularly those that depend on JavaScript features like providing anonymous functions as arguments to Hydra functions. The task of extending the model to provide for this greater level of abstraction, while preserving some modicum of protection against infinite computations, remains an open (and large) one.

Memorias is a performance piece by Jessica Rodríguez that contains six esoteric languages (Escribir, Observar, Leer, Oir, Ver, Escuchar) or "esolang" [19]. Each language is built over different audio-visual live coding languages (using Estuary's JSoLang feature, see below), and their syntax is inspired by six autobiographical stories. This produces valid statements that can be read as poetry but that also trigger sonic or visual events. In the current version of the piece, one of the esolangs (Oir) is now parsing Hydra's functions that focus on screen feedback.

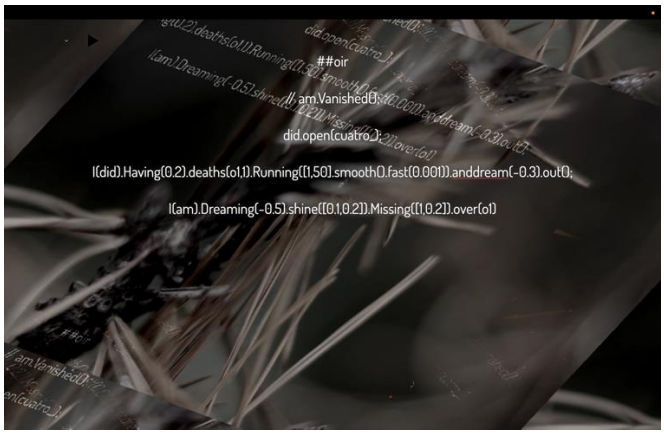

Figure 3. A JSoLang over Hydra, from *Memorias* by Jessica Rodríguez.

3. TOWARDS A MODULAR SANDBOX

Estuary is a project that brings together considerable human and material resources over a period of seven plus years, in the context of specific territories and a web of relatively specific working and artistic relationships. One of the responsibilities that comes with this concentration of resources is to make sure the results of the project are as useful as possible to other people beyond that web of relatively specific relationships (but within the basic definition of what Estuary "is", still). For this reason, there is a priority placed on making Estuary extensible and customizable in various ways, and work on this extensibility and customization has become a second emphasis of recent years (alongside the abovementioned shift to audiovisual rather than musical live coding).

The earliest versions of Estuary had support for visual customization via CSS themes, the capacity to flexibly define the interfaces used by a particular ensemble (more about this below), and limited support for localization of the natural language used throughout the interface. In recent years, work has proceeded on

more fundamental forms of extension of the software, which are described in the subsections to follow, and include a system for adding web-located audio samples on the fly, a system for describing – also on-the-fly – transpiled languages "over" other already existing languages, and a proliferation of the types and appearances of the available "widgets" that can be included in Estuary ensembles (with a particular focus on those that are useful for intra-ensemble communication).

Fundamentally, the goal of this work on making Estuary as extensible as possible is to facilitate others making things with the software that are radically unknown and unforeseen by the research team and lead maintainer, without requiring a relationship or agreement or collaboration (as would be the case for things that are "built-in" to Estuary). Even more fundamentally, this priority is about an orientation to an unconditional and uncertain (and always unguaranteed) state of hospitality [20], as an alternative to aiming for a solipsistic specificity or an unattainable (and authoritarian) universality.

3.1 On-The-Fly Media Resources

One form of extensibility added to Estuary in recent years is a system whereby references to web-located audio samples can be added "on-the-fly" and then used by languages that use WebDirt to produce "sample playback events" (eg. MiniTidal, TimeNot, Seis8s). It works as follows: each ensemble has a (collaboratively editable) list of resource operations. One type of resource operation adds an association between a URL (eg. where a sound file "lives" on the Internet), and a "tuplet" consisting of a name (like "bd" or "gtr") representing a samplebank, and a number representing a specific sample within that bank. Another type of resource operation adds a whole list of such associations from an Internet-located JSON file (a "reslist" - short for "resource list"). Other resource operations can manipulate the map of available resources in various ways (providing aliases, removing entries, etc).

On the basis of this system, it is possible to curate collections of audio samples at specific Internet locations, storing a "reslist" at the same location as the samples. A terminal script is provided to make it easier to auto-generate reslists in this common circumstance. At that point, a particular "curated" collection of samples can be added, with a single operation in Estuary's terminal, and a particular ensemble may, on a moment-to-moment basis curate both the samples that are available and – characteristically for live coding where the way in which things are named is especially significant – how they are called. Currently these operations are completed in Estuary's "terminal" (an interface not unlike a UNIX shell where commands are typed are textual responses come back) but the intention is to provide a graphical interface for these operations, as well as expanded support for scanning and exploring "community" collections of samples before adding them to a specific ensemble context. The Estuary wiki (at the project's main github repository [21]) currently contains a growing list of sample collections ("reslists"), most licensed on a CC0 basis (public domain), that can be immediately added to an ensemble and explored.

One initiative that has already used the on-the-fly resource system heavily is the weekly WeekEndJam (WEJ) event, wherein the sharing and collective exploration of newly available sample resources is a common occurrence. The WeekEndJam grew from a desire between three Tidalcycles forum users to play music/sounds/visuals together remotely, during the height of the COVID pandemic/lockdowns in January 2021. WEJ has since enjoyed the patronage of between 30-50 individual live coders, including people interested in very different styles of music/art, and with very different durations and histories of engagement with live coding. A public invitation is sent out each week shortly before starting (on the Estuary Discord server, among other places), and there is rarely ever any plan for the hour-long session. All sessions are streamed live, recorded for review and highlights extracted/shared for enjoyment. Often the group will explore new platform features, such as new sample resources that someone has made available, updates to the available languages, or JSoLangs (see below). WEJ organizer Bernard Gray notes: "Personally I've enjoyed the exploration of collaborative free improvisation in a live coding context, working beyond audio and into visual spaces as well. When everyone in the band can play every instrument all at the same time (including the projector), it's fascinating to see how different people respond to, and work within, that freedom."

The "resources" that people may wish to curate into lists that can be used in Estuary's collaborative ensemble spaces are not limited to sound resources. Extending the on-the-fly media resource system to video and image files is an imminent next step in this work, but not the only way in which this system could be extended. 3D models could be such resources (see discussion of LocoMotion, below), for example. Another possibility is to start to include curated (and searchable) libraries of example programs, akin to the "Giblets" that were an early and key form of code sharing in Gibber [22].

3.2 JSoLangs

Another recent form of extensibility that has been added to Estuary is a system for languages that are defined on-the-fly and transpiled into other languages supported by the system: JSoLangs (pronounced Jay-Esso-Langs, a portmanteau of JS for JavaScript and esolang, for "esoteric programming language"). Estuary's JSoLang system [6] uses the Peg.js library for writing recursive descent parsers using JavaScript. To define a JSoLang, one enters a Peg.js parser in one of Estuary's code editing panels, with an initial header that identifies the name of the new JSoLang. Thereafter, when code that is identified as being "in" the JSoLang is evaluated, the Peg.js parser [23] is used to transpile the provided text into new text code (including, as part of that output, Estuary's specific pragmas for identifying a language with which the newly provided text should be evaluated).

A particularly simple and frequent application of the JSoLang system has been as a way of tweaking the lexicon of an existing language, providing lists of translations from specific strings of characters to other strings of characters. This can support specific poetic purposes (a way of manipulating the language the programmer thinks in, and what is shown to an audience) as well as, more generally, opening up simple pathways for working with different alphabets included in the Unicode standard. Earlier work with Estuary had demonstrated the felicity of creating new languages over existing languages in brief workshop settings [24]. That work involved laboriously incorporating new modules into, and rebuilding, the Estuary client itself – now, with the JSoLang system, such "workshop languages" can be created (and maintained) much more fluidly. Here is an example of how a simple "text replacement" JSoLang is defined, using peg.js:

```
##JSoLang text-replacement-example
main = x:allRules* { return "##tidal\n" + x.join("") }
allRules = bassdrum / clap / anyCharacter
anyCharacter = .
```

```
bassdrum = "bassdrum"i { return "bd" } // bassdrum will be replaced with bd
clap = "clap"i { return "cp" } // clap will be replaced with cp
```

At the time of writing, members of the SuperContinent ensemble, established in 2018 to explore and benefit from Estuary's utility for geographically distributed live coding, have begun to explore the possibilities inherent in using JSoLangs to live code together with diverse alphabets. The ensemble currently has eight members from Japan, Canada, India, UK, Colombia, Portugal, and South Africa who rehearse weekly, exploring and developing inclusive, non-hierarchical collective-live-coding practices.

Studio//Stage, by Kate Sicchio, is another example of an early application of Estuary's JSoLang system. Studio//Stage is transpiled into CineCer0 code (to control the playback of videos with precise relationships to time), but the language itself uses choreographic terminology to code video timings, loops and playback functions. The aim is to craft movement in the "studio" while also developing the language, and then to use this vocabulary on the "stage" during performance. For example, the slowing down of a video clip to a certain framerate may make the movement appear heavy. This new slow framerate is now named 'heavy' and is referred to as such during a live coding performance. Studio//Stage allows for algorithmic choreographies to emerge on the screen as if a choreographer was working with dancers in the studio, developing movement phrases and materials.

3.3 Widgets for Intra-Ensemble Communication

A third significant form of extensibility in Estuary is the ability to define custom interfaces. Estuary's "View system" has supported such custom "views" of "what is happening" in an Ensemble since its earliest stages. Recent work has focused on proliferating the available widgets that can be included in such views, with an emphasis on widgets useful for intra-ensemble communication. This work is still in an in-progress/alpha stage and has not yet been used much in practice, so this section will be somewhat briefer than others.

Examples of new widgets that are already available in Estuary (while continuing to be refined on an ongoing basis) include: notepads to which pages can be added and on which ensemble members can share notes about strategies, concepts, available resources, etc; calendar widgets which allow information about upcoming meetings, workshops, or rehearsals to be edited by ensemble members, with date and time displayed both in local and universal terms (an important practical consideration for geographically networked ensembles); widgets that visualize the prevailing tempo (ie. metric grid) in the ensemble in different ways; widgets that mark the passage of time by counting down (in different ways) from preset times, or by counting up from a starting time; widgets that visualize the computational demands made by the current code; widgets that facilitate collaborative turn-taking in ensembles, such as a "roulette" widget designed around the common live coding practice of roulette (wherein performers take turns modifying the same code); widgets that augment Estuary's longstanding "ensemble-wide" chat system (displayed in a collapsible terminal at the bottom of the screen) with specific, separate chats that are displayed in specific locations within a View (and which might be used, for example, to facilitate conversations among sub-groups of those participating in an ensemble,cf. the suggestion of [25] that embedding chat affordances directly adjacent to shared code could be very helpful in pair/trio programming and turn-taking). It is also possible to embed arbitrary web pages as widgets, which can be used, among other things, to embed web-based video-conferencing applications within what is displayed, potentially sharing webcam views of performers with each other, or sharing generative visuals to computers lacking the computational power (eg. a good graphics card) to render them.

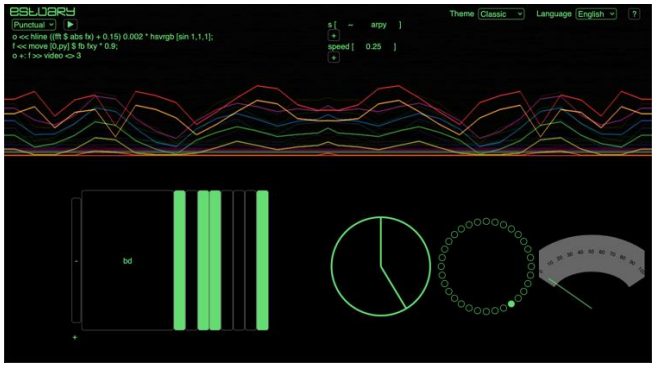

Figure 4. A selection of new and old Estuary widgets

We are also beginning to add alternative formats/display behaviours to Estuary's core "code editor", and see this as a particularly fruitful area for further development (code, particularly but not only in live coding, being already a medium for communication between collaborating humans). The precedent demonstrated by collaborative live coding environment Troop [26] shows the promise of using text colouration and moving icons to provide a visible trace of the activity and presence of specific individuals in a collaborative live coding ensemble – Estuary could evolve to provide this behaviour as an option, and then also explore other ways of making individual activity legible in the potentially opaque space of screen-based interactions.

An Estuary ensemble contains any number of "Views" which are definitions of which widgets to display. In the most common cases, all clients connected to a particular ensemble will display a "default" view that has been established for that ensemble. However, it is not necessary for all clients to display the same View/interface. The same ensemble can be "viewed" by different Views at the same time – a feature which supports a number of interesting live coding and network music configurations. For example, an "audience view" might be composed around specific decisions regarding what to share with an audience, while specific performers in an ensemble might use different views that give them the ability to intervene in the results that are generated (and in the code that is shared with an audience) while also facilitating consultation with notes and help, "backchannel" discussions or decision making processes, tracking of computational load, compositional plans, etc – all of which may not always need, or be desired to be, shared with an audience. In another common application of this flexible, multiple View system, a View without any widgets might be used to display generative visuals completely separately (ie. with a separate computer and projector/display) from the code that generates those visuals, addressing the common challenge of being able to read code and appreciate large format generative visuals at the same time.

4. FURTHER APPLICATIONS

In this section, we provide some accounts of interesting situations in which Estuary has been used, many of which have a connection to different kinds of educational settings.

4.1 Zero-installation live coding workshops

Estuary's provision of a multi-lingual live coding environment, in which all languages are "ready to go" without the installation of software beyond the web browser itself, is a significant advantage in many workshop situations. Since 2021, Antonio Roberts has been delivering regular live coding workshops at Birmingham Open Media (BOM) in Birmingham, UK, as well as workshops about Blender, Pure Data, and Imagemagick. When approached about delivering live coding workshops, there was excitement but also slight hesitation due to the difficulty of installing common live coding software packages, which – even when they offer "easy" installation via single-line terminal commands – still rely on knowledge of the terminal and the skills needed to fix bugs if the installation fails somehow. Estuary made this process a lot easier as all of the software was available in the browser. This meant easier installation and setup at BOM (all of their computers had a browser already installed) and workshop participants could easily practice what they had learnt at home. The workshops were part of the Pulsar Summer Camp for up to 15 children aged 7 -12. The first workshop focused on making music using Tidal/MiniTidal and the second day focused on making visuals using Hydra. Using Estuary provided a common interface which all could work from, while still allowing for flexibility and customization. Having a common, collaborative interface helped when communicating instructions and also bug fixing/addressing questions as the answers could be supplied to everyone. Having one interface for both Tidal and Hydra also provided consistency over the two workshops which meant less time learning new interfaces. [27,28,29,30]

4.2 Laptop ensembles

For the past two years, Carl Testa has used the Estuary platform in the context of a burgeoning electroacoustic laptop ensemble at the high school level, in New Haven, Connecticut, USA. Estuary's integration of both TidalCycles (in the form of MiniTidal) and Hydra Video Synth solved many problems at once in this educational context. Students being able to see each other's code contributions meant beginning students could easily learn from more advanced students. In the context of hybrid learning, being able to easily incorporate students learning from home with students learning in class was an immediate boon to the work in class. Finally, having code being executed and shared across of the ensemble instead of audio being sent solved issues of audio latency. For the past year, the work in the class has often been focused on learning new commands, parameters, and syntax of the MiniTidal language and then incorporating that knowledge into improvising as an ensemble. They then review their improvisations and reflect on moments or strategies that seemed to work compositionally, finding ways to incorporate that into future practice. [31]

Carl Testa notes: "I've found that Estuary encourages and enables improvising with orchestration in a way that helps students become aware of the various roles they can take on in an ensemble context. Since everyone is playing the same instrument and can contribute to the music from a plurality of perspectives, methods, and sound sources, the students must make choices in the moment that enable them to "find their space in the music". They must adjust their timing, pitch and register selection, and timbral choices so that they can hear their contributions and make sure their contributions are distinct and provide space for everyone to contribute. We are now taking some of what we've learned about improvisation and incorporating it into an electroacoustic improvisational context incorporating trumpet, guitars, Estuary, and live processing with Sonic Pi."

4.3 Estuary as an instrument instead of an environment

The Concordia Laptop Orchestra (CLOrk), a large ensemble of 25-30 members, utilizes Estuary most often in combination with other audiovisual creations, conceptualizing Estuary as one instrument among others in the orchestra rather than (as is more common) the entire environment or medium for the orchestra's performance. At times, Punctual code on Estuary has been used as an audio-responsive visualization tool in telematic performances, in which the distributed orchestra's diverse sound (generated by various digital, analog, and acoustic instruments and sometimes vocally) has been collected on one computer and then routed to Estuary via the browser's microphone input. In such configurations, visuals were live-coded in an Estuary ensemble window and shown to audiences in another (code-invisible) instance of the ensemble, thus utilizing the Estuary ensemble as a means for visual-performance remote-control. A recent example of this can be seen in CLOrk's movement "Reaching the Karman Line" in CIMaCC's Nature Minds! piece at Cambridge Festival 2022. Estuary has also sometimes been used to set up standalone unattended laptop performers, where pre-coded algorithms generated sounds as part of a CLOrk performance. In these instances, the Estuary ensemble was similarly used as an invisible, easy-access remote-controllable smart orchestra instrument, not a collective live coding performance environment. A recent example of this usage can be seen in Act 4 of lost connection, a mini-opera in collaboration with the RISE project based on an idea by Juanita Marchand Knight, in which laptops algorithmically were to converse with each other as independent futuristic artificial beings.

4.4 Role-playing, world-building, gameplay

Role-playing and narrative design gives users the opportunity to learn and experiment outside of their normal perception. Similar to how language is fundamental to human culture, world building in combination with role playing can be manifested through the Estuary platform, by allowing different languages to signify different aspects of the world while actively being both performed and edited on the fly. This can be used in genres such as sci fi and Afrofuturism. In February 2021, Kofi Oduro demonstrated this aspect of Estuary for an Afrofuturism class at Tulane University. Users were encouraged to engage with a world-building exercise as inhabitants of external planets. Each panel in a predefined Estuary View was treated as a different planet or house, and users were prompted verbally and textually to either add, remove or remix the code in that panel to match the changes in this jointly created world. Elements and symbolisms in this particular experience matched the theme of the class.

Role-playing can also be extended through gameplay as observed in the coding performances of UrTeam (Barry Wan, Shaun Bellamy, Tareq Abu-Rahma and D.Andrew Stewart), a Canadian-Czech-Hong Kongese-Palestinian quartet that embraces Estuary's multi-language features for both sound and visual coding. Inspired by the tradition of improvisation to silent film, UrTeam often presents their own unique sound and music design to video game

sequences and in some contexts, merges the visual coding environment of Estuary with live game play. The approach of UrTeam presents interesting creative challenges that require an understanding of using sound and music to elaborate and enhance narratives conveyed through visual media.

5. FUTURE WORK

Some aspects of future work on Estuary involve continuing to proliferate options and extend capabilities along the lines discussed above in sections 2 and 3: more widgets that support specific types of communication, visualization, or decision-making in ensembles; further evolution in terms of what is supported by the restricted number of "base" live coding languages; further work on modular, on-the-fly systems for incorporating media resources beyond sound samples (such as the images and videos "consumed" by languages like Punctual, CineCer0, and Hydra) and continuing work on attempting to reduce the computational demands of Estuary within the browser context.

The computational demands of Estuary are especially salient (and challenging) when it is used with larger ensembles, a general characteristic of systems that involve everyone's code being rendered on everyone's computer [32]. Laptop orchestras and live coding workshop games might involve ten, fifteen, or more performers simultaneously working on different audiovisual layers. The demands of generative visuals in Estuary tend to be manageable even with larger numbers of performers, assuming the presence of relatively contemporary accelerated graphics processing units (GPUs). The demands of generative audio, however, rendered by the Web Audio API, appear to be a more common source of "computational stress", particularly when Estuary is used with larger ensembles. Many audio results in Estuary are rendered by the WebDirt library, which uses the Web Audio API to play back samples using (mostly) built-in audio nodes of the Web Audio API. The computational demands of this sample playback engine scale more or less linearly in proportion to the number of sample events that are present in any given slice of time, a number that can grow quite rapidly if many people in an ensemble are triggering events often and/or are triggering events that last for longer durations. Nonetheless, it appears that, in the current configuration of the Web Audio API, this computational work is performed in a single thread running on a single core of the underlying machine (leaving the few or many additional cores that the machine may have as an untapped resource). Large ensembles using Estuary often need to adopt and rehearse strategies to manage this computational demand, such as preferring shorter samples, or figuring out through trial and error how musically "busy" the ensemble can afford to be. Moreover, this computational limit moves downwards as less powerful machines are used, a factor which simultaneously makes the platform unacceptably dependent on "cutting edge" hardware (and the cycles of technological obsolescence and waste connected to the availability of such hardware), while also excluding people with less privileged access to such hardware. One line of investigation in response to this challenge would be to reimplement WebDirt without using the standard Web Audio nodes, in favour of doing all sample playback and processing in an AudioWorkletNode that can run on an additional available processing core and could be implemented using WebAssembly. Another line of investigation would be to produce an alternate machine-code build of Estuary's client, while simultaneously continuing to build and maintain the browser-based version of the client, following a pattern that is also seen with many mainstream collaboration and telecommunication platforms, and taking advantage of the fact that the reflex-platform, used to build Estuary's client, is already set up to compile to machine code targets. At the time of writing, this remains an open question for the project.

A second major direction in which future work on Estuary is expected to go involves enhanced possibilities for modular relationships even with what might be thought of as "base" languages. Hitherto, the base languages in Estuary have involved substantial Haskell codebases that are compiled at the same time as the rest of the Estuary client. This requirement to build Estuary AND its languages at the same time works against modularity. To address this, a new standard for Estuary-oriented "ExoLangs" (external or exoteric languages, another pun on EsoLang), is currently in active development, and allows base languages to exist as compiled JavaScript code that is "linked" into an ensemble on the fly, just as media resources and JSoLangs currently are. For the new language/project, LocoMotion [33], initiated by David Ogborn and Kate Sicchio, that is intended to exist both within Estuary and separately, the language is being built with PureScript so that the compiled JavaScript which results can be added to Estuary in just this way, as an ExoLang. This will allow new versions of the new language (which are expected to come fast and furiously) to be used in Estuary without the rather laborious process of rebuilding and redeploying the entire platform. This approach will also enable people to add their languages to an Estuary setting without the need for discussion and collaboration with either the Estuary research team or the host of a particular Estuary server – a kind of unconditional, uncertain hospitality.

6. ACKNOWLEDGMENTS

The development of Estuary has been supported directly by two grants from the Social Sciences and Humanities Research Council (SSHRC) of Canada, "Projectional editing for musical live coding" and "Platforms and practices for networked, language-neutral live coding." The development of LocoMotion is supported by an Exploration grant from the New Frontiers in Research Fund (NFRF). The Networked Imagination Laboratory at McMaster University, funded by the Canada Foundation for Innovation (CFI) and Ontario's Ministry for Research and Innovation, has been a key environment for development and application and outreach activities related to Estuary. Profound thanks to all of the people who have made Estuary a part of their explorations of live coding, including but not limited to all of the members of the collaborative ensembles mentioned in this text, as well as the Cybernetic Orchestra at McMaster University (which has often been the first place that new collaborative Estuary features are tried out).

7. REFERENCES

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

[2] McLean, A. (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. New York, NY, USA, 63–70. ACM Press. doi:10.1145/2633638.2633647.

[3] Tanimoto, Steven L. 1990. “VIVA: A Visual Language for Image Processing.” *Journal of Visual Languages & Computing* 1 (2): 127–39.

[4] Tanimoto, Steven L. 2013. “A Perspective on the Evolution of Live Programming.” In *1st International Workshop on Live Programming (Live)*, 31–34. IEEE.

[5] Jacques Derrida (1998). *Monolingualism of the Other: or, The Prosthesis of Origin*. trans. Patrick Mensah. Stanford: Stanford University Press.

[6] David Ogborn, Clarissa Littler, Kate Sicchio (2021). "JSoLangs: ephemeral esolangs in a collaborative live coding environment." Proceedings of *CSDH-SCHN 2021: Making the Network*. https://hcommons.org/deposits/download/hc:39064/CONTENT/jsolangs-paper-csdh-2021.pdf/

[7] Graham Hutton and Erik Meijer (1996). *Monadic Parser Combinators*. Technical report NOTTCS-TR-96-4, Department of Computer Science, University of Nottingham

[8] https://github.com/dktr0/MusicW

[9] https://github.com/dktr0/WebDirt

[10] https://github.com/dktr0/haskellish

[11] Benjamin H. Bratton (2016). *The Stack: On Software and Sovereignty*. MIT Press.

[12] Wendy Hui Kyong Chun (2017). *Updating to Remain the Same: Habitual New Media*. MIT Press.

[13] https://dktr0.github.io/Punctual

[14] Julian Rohrhuber, Alberto de Campo, and Renate Wieser (2005). "Algorithms Today: Notes on Language Design for Just in Time Programming." *Proceedings of the International Computer Music Conference*.

[15] Smith, R. Lawson, S. (2018) “Rogue Two: Reflections on the Creative and Technological Development of the Audiovisual Duo—The Rebel Scum.” *Journal of Electronic Dance Music Culture* 10:1, pp. 63-82, DOI: https://dj.dancecult.net/index.php/dancecult/article/view/1026/946

[16] Jessica Rodríguez, Esteban Betancur, and Rolando Rodríguez (2019). "CineVivo: a mini-language for live-visuals." *Proceedings of the International Conference on Live Coding*.

[17] https://hydra.ojack.xyz/

[18] Turing, A. M. (1937). "On Computable Numbers, with an Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*. Wiley. s2-42 (1): 230–265. doi:10.1112/plms/s2-42.1.230.

[19] Daniel Temkin (2015). "Esolangs as Experiential Art". *Proceedings of the International Symposium on Electronic Art* (ISEA).

[20] Jacques Derrida (2000). *Of Hospitality* (Anne Dufourmantelle invites Jacques Derrida to respond). Trans. Rachel Bowlby. Stanford University Press.

[21] https://github.com/dktr0/Estuary

[22] Charlie Roberts, Karl Yerkes, Danny Bazo, Matthew Wright, JoAnn Kuchera-Morin (2015). “Sharing Time and Code in a Browser-Based Live Coding Environment.” *Proceedings of the International Conference on Live Coding*, Leeds, UK.

[23] https://pegjs.org

[24] Luis N. Del Angel, Marianne Teixido, Emilio Ocelotl, and David Ogborn (2019). "Bellacode: localized textual interfaces for live coding music." *Proceedings of the International Conference on Live Coding*.

[25] Anna Xambó, Gerard Roma, Pratik Shah, Takahiko Tsuchiya, Jason Freeman, Brian Magerko (2018). “Turn-taking and Online Chatting in Co-located and Remote Collaborative Music Live Coding.” *Journal of the Audio Engineering Society* 66:4, pp. 253-66.

[26] Ryan Kirkbride (2017). “Troop: A Collaborative Tool for Live Coding.” *Proceedings of the 14th Sound and Music Computing Conference*, Espoo, Finland.

[27] https://bom.org.uk/engagement/for-kids/summer-camp/pulsar-summer-camp-2021/

[28] https://bom.org.uk/how-to-code-live-music-beginners/

[29] https://bom.org.uk/engagement/for-kids/summer-camp/

[30] https://bom.org.uk/how-to-code-live-art-advanced/

[31] https://youtu.be/Jum75_iIXQc

[32] Charlie Roberts, Ian Hattwick, Eric Sheffield, and Gillian Smith (2022). “Rethinking networked collaboration in the live coding environment Gibber.” *Proceedings of the International Conference on New Interfaces for Musical Expression*, Waipapa Taumata Rau, Aotearoa.

[33] https://dktr0.github.io/LocoMotion