**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

Programa de Maestría y Doctorado en Música

Facultad de Música

Instituto de Ciencias Aplicadas y Tecnología

Instituto de Investigaciones Antropológicas

# SonoTexto:

# Relación entre práctica artística y desarrollo tecnológico en el live coding

Tesis que, para optar por el grado de Doctor en Música (Tecnología Musical)

Presenta

Hernani Villaseñor Ramírez

Tutor principal

Dr. Jorge David García Castilla (Facultad de Música, UNAM)

Miembros del comité tutor

Dra. Paz Sastre Domínguez (DAH/DCSH, UAM unidad Lerma)

Dr. Iván Paz Ortiz (BarcelonaTech)

Ciudad de México, octubre 2022

Declaro conocer el Código de Ética de la Universidad Nacional Autónoma de México, plasmado en la Legislación Universitaria. Con base en las definiciones de integridad y honestidad ahí especificadas, aseguro mediante mi firma al calce que el presente trabajo es original y enteramente de mi autoría. Todas las citas de obras elaboradas por otros autores, o sus referencias, aparecen aquí debida y adecuadamente señaladas, así como acreditadas mediante las convenciones editoriales correspondientes.

**Agradecimientos**

Quiero agradecer el tiempo y dedicación de todas las personas involucradas en la realización de este trabajo, quienes generosamente lo discutieron, leyeron, comentaron y gestionaron en sus diferentes etapas.

De manera particular quiero expresar mi agradecimiento a Jorge David García Castilla por aceptar el reto de dirigir esta tesis; a Paz Sastre Domínguez y a Iván Paz por integrarse al comité tutor y nutrir el trabajo con sus comentarios, lecturas y recomendaciones; a Iracema de Andrade, Juan Sebastián Lach Lau y Hugo Solís García por sus lecturas y comentarios durante la candidatura y el proceso de graduación; a Frank Dufour por su lectura y comentarios durante el XIV coloquio de alumnos del posgrado; al Seminario Doctoral, dirigido por Roberto Kolb, Lizette Alegre y Jorge David, quienes junto a mis compañeros y compañeras escucharon, comentaron y compartieron diferentes trabajos de investigación musical; a las y los integrantes del Seminario Permanente de Tecnología Musical por las presentaciones, discusiones y proyectos realizados en conjunto.

A los equipos que organizaron CAC.6 2018, Re:Sound 2019, ICLC 2020 y los Coloquios XIV y XV de Alumnos del Posgrado, quienes proporcionaron las condiciones y los espacios para exponer las ideas y los avances de mi trabajo doctoral.

A mi familia, en especial a Josefina y Heriberto, quienes estuvieron siempre apoyando y al tanto; a Sabine, quien me acompañó y compartió de cerca el proceso del doctorado; a Isolda por la revisión de redacción del documento.

A las desarrolladoras y desarrolladores de software y lenguajes de programación libres, quienes comparten el código fuente en diferentes plataformas. A la comunidad de live coding que, a través de sus practicantes, exploran la creación musical con lenguajes de programación.

Asimismo, agradezco a todas las personas que integran y coordinan el Programa de Maestría y Doctorado en Música de la UNAM por gestionar todo lo necesario para llevar a cabo el presente trabajo de doctorado y al CONACYT por otorgarme una beca para realizarlo.

# Índice

# Índice de figuras

## Índice de tablas

# Introducción

Son cerca de las cinco de la tarde del 27 de enero de 2011. Varias personas se encuentran sentadas en el piso de la Galería Manuel Felguérez del Centro Multimedia del Cenart[1] en la Ciudad de México mientras esperan el inicio de un concierto de música e imagen con computadoras. El escenario, algo improvisado, se encuentra en el centro de la galería al nivel de la audiencia. Sobre una mesa hay dos computadoras portátiles conectadas a un sistema de sonido y a dos proyectores que emiten su imagen hacia una pared de la sala. En algún momento se anuncia que la sesión está por comenzar y que la presentación que van a presenciar es una práctica de programación en vivo para crear música e imagen conocida como *live coding*. Enseguida dos personas se colocan frente a su respectiva computadora y comienzan a escribir código. Sobre la pared de la galería se proyecta una réplica de la pantalla de cada computadora que muestra el proceso de escritura de símbolos, números, letras, caracteres y palabras estructuradas en el código fuente que cada participante teclea. Unos segundos después comienzan a escucharse y a verse los primeros sonidos e imágenes que resultan de la escritura y ejecución del código fuente que la dupla programa con lenguajes de programación desarrollados para fines creativos. Por las bocinas suena una música compuesta por sonidos sintéticos, ruidosos y organizados en ciclos, mientras que en la pared varias formas geométricas se mueven. A veces no todo marcha bien, durante este proceso hay fallos en la programación y lo que suena o se visualiza deja de cambiar, se detiene o incluso nunca llega a producirse. Las pantallas proyectadas también muestran los errores del código fuente cuando se generan, en ese momento alguien se acerca a sugerir una posible solución y desde la audiencia surgen muestras de apoyo cuando los errores son superados y el sonido o la imagen vuelven a generarse. Algunos minutos después el dúo termina su presentación e inmediatamente otras dos personas ocupan la mesa del escenario con sus computadoras portátiles, intercambian cables de audio e imagen con la pareja anterior y repiten una dinámica parecida: alguien programa sonido y alguien programa imagen.

1 El Centro Multimedia del Centro Nacional de las Artes fue creado por el CONACULTA en el año 1994 “con el objetivo de impulsar el uso de nuevas tecnología en la creación artística” (Sáizar, 2009, p. 5).

De esta manera el concierto transcurre con la participación de varias duplas y al final se anuncia la próxima sesión de live coding que se llevará a cabo el siguiente mes.

*Figura 1. Sesión 1 de Live Coding en la Galería Manuel Felguérez del Centro Multimedia, 27 de enero de 2011. Izquierda: equipo y escenario para la sesión. Derecha: Furenku programa sonido y Mitzi Olvera imagen. Archivo personal del autor.*

Si bien la presentación descrita sucedía a inicios del año 2011, llegar a ese momento fue el resultado de un conjunto de actividades de enseñanza y difusión de la música y el arte computacionales. Tales actividades tienen su origen alrededor del año 2006 en el Taller de Audio del Centro Multimedia (Cmm) y se desarrollaron durante cinco años hasta formar la primera comunidad de live coding[2] en México en el año 2011 que estaría activa en ese espacio hasta el año 2014. De manera específica, el entonces Taller de Audio era el encargado de promover diversas prácticas de arte digital relacionadas al sonido entre las que se encontraba el live coding.

Dicha práctica es una forma de creación artística con lenguajes de programación surgida en el Reino Unido, Alemania y Estados Unidos a inicios de los años 2000, la cual consiste en programar una computadora durante un concierto, presentación o performance para generar música, imagen o alguna otra salida artística como danza o literatura electrónica. La primera referencia de un concierto de live coding en México aparece fechada el 26 de octubre de 2006 en

2 El término *live coding* lo reconozco como un anglicismo, sin embargo, sigo la convención propuesta por Chiantore, Domínguez y Martínez (2016), quienes sugieren escribir sin cursivas las palabras extranjeras que pertenecen al vocabulario de una tradición musical.

la wiki de la organización promotora de live coding Toplap[3]. El concierto fue realizado por la banda de computadoras mU[4], cuyos integrantes Ernesto Romero, Ezequiel Netri y Eduardo Meléndez formaban parte del Taller de Audio junto a Katya Alvarez.

Desde entonces se puede rastrear en la sección de *Actividades Anteriores* del sitio web del Cmm[5] y del sitio web del Taller de Audio hospedado en la aplicación Google Sites[6] la serie de cursos, conciertos y encuentros que propiciaron el terreno que consolidó la comunidad de practicantes de live coding mencionada. Tal comunidad se formó en el marco de una serie de presentaciones mensuales llamadas *Sesiones de Live Coding* realizadas en las instalaciones del Cenart y en otras instituciones que colaboraron en su organización entre enero de 2011 y julio de 2014. La participación en dichas sesiones estaba conformada por asistentes a cursos impartidos en el Cmm, integrantes del Taller de Audio o personas que recibían una invitación. Para ello había una convocatoria publicada en el sitio web del Taller de Audio que se replicaba mensualmente en una lista de correos electrónicos recabados al final de las sesiones.

La programación durante las sesiones se realizaba partiendo de cero, modalidad conocida como live coding *from scratch*, en un tiempo de 9 minutos. Esa dinámica motivó a la comunidad a utilizar estrategias de memorización y escritura de código fuente para llevar a cabo las presentaciones[7]. Aunque esta modalidad tiene su origen en el inicio del live coding practicado en Europa y Estados Unidos, la forma de organizar las sesiones en participaciones de 9 minutos por parejas es una aportación del Taller de Audio. Lo anterior se observa en sesiones realizadas posteriormente por otras comunidades como el colectivo Toplap Barcelona, cuyos integrantes han adaptado la dinámica iniciada en México a su forma de entender el live coding (Villaseñor y Paz, 2020).

Mi vínculo con el Taller de Audio fue como alumno de diversos cursos, entre los años 2008 y 2009 y, posteriormente, como integrante de ese espacio entre los años 2010 y 2014.

3 Véase la página web ToplapEvents de la wiki del colectivo Toplap: https://toplap.org/wiki/ToplapEvents.
4 Véase acerca de mU: https://sites.google.com/site/tallerdeaudio/mu.
5 Véase: http://cmm.cenart.gob.mx/Memoria#ActividadesAnteriores.
6 Véase: https://sites.google.com/site/tallerdeaudio/actividades/actividades-anteriores.
7 La primera sesión de live coding, según la sección de ‘actividades anteriores’ del sitio web del Centro Multimedia Cenart, se realizó el 27 de enero de 2011. La información del evento dice que cada participante contaba con 15 minutos para programar, aunque en las siguientes sesiones este tiempo pasó a 10 minutos y finalmente se estableció en los 9 minutos que las caracterizaron durante los siguientes cuatro años.

Cuando trabajé ahí colaboré durante cinco años con sus integrantes y con las personas que realizaban su servicio social –en ese entonces, y en orden cronológico, Ernesto Romero (jefe del Taller hasta 2013), Rodrigo Guzmán, Juan Pablo Méndez, Luis Navarro, Emilio Ocelotl, Ivonne Valdez (servicios sociales) y Alberto Cerro (jefe del Taller hasta 2015)– para impulsar la práctica de live coding como gestor, enseñante y *live coder*.

Como mencioné antes, la comunidad de live coding formada en el Cmm se mantuvo activa dentro de esa institución hasta finales de 2014 cuando las Sesiones de Live Coding y otras actividades relacionadas a esta práctica llegaron a su fin. Esto sucedió debido a que quienes integrábamos el Taller de Audio en ese momento dejamos de trabajar ahí por cambio de residencia, inicio de estudios de posgrado o finalización del servicio social.

Después del año 2014, el live coding en México ha continuado su desarrollo y ha sido acogido por diferentes personas, en su mayoría con algún vínculo con el Cmm, provenientes de diversos campos como la música, el arte, las humanidades y las ciencias. Entre ellas se encuentran Ernesto Romero, Luis Navarro, Libertad Figueroa, Rodrigo Frenk, José Carlos Hasbun, Iván Paz, Alexandra Cárdenas, Ofelia Negrete, Emilio Ocelotl, Alejandro Franco, Jessica Rodríguez, Marianne Teixido, Malitzin Cortés, Mitzi Olvera, Diego Villaseñor de Cortina, Jaime Lobato y Dorian Sotomayor. Asimismo, las bandas de laptop y colectivos Radiador, MicoRex, LiveCodeNet Ensamble, RGGTRN, PiranhaLab y Toplap Mx.

Al igual que otras comunidades de live coding surgidas en diferentes partes del mundo, la comunidad de México se integró desde sus inicios a una práctica global ligada a lenguajes de programación creativa, plataformas tecnológicas, software y organizaciones. En el caso de México se puede mencionar una primera aproximación al live coding con los lenguajes de programación SuperCollider, Fluxus, Processing, Pure Data, VVVV y la librería de C++ openFrameworks, así como la afiliación al colectivo Toplap. Más tarde, comenzarían a utilizarse los lenguajes de programación dedicados al live coding TidalCycles e Hydra e iniciaría una afiliación al movimiento Algorave[8].

8 Algorave es la combinación de las palabras algoritmo y rave que hacen alusión a fiestas donde se toca música electrónica para bailar con el uso de algoritmos. Véase: https://algorave.com/.

*Acerca de la problemática*

A unos años de distancia del inicio de la comunidad de live coding del Cmm reconozco que esta iniciativa generó conocimiento, impulsó carreras artísticas y motivó investigaciones académicas en el contexto de posgrados nacionales e internacionales. Sin embargo, aunque en ese entonces había entusiasmo y motivación entre los participantes de la comunidad para aprender y compartir conocimientos técnicos del uso de lenguajes de programación dedicados a la creación artística, tal iniciativa carecía del empuje interno del Taller de Audio y de las condiciones para reflexionar de manera crítica sobre la práctica artística del live coding y las tecnologías empleadas. En ese sentido, el uso de los lenguajes de programación se hacía desde una perspectiva utilitaria que no incentivaba el diseño de software y tampoco el análisis del contexto bajo el que eran empleados.

Si bien el Taller de Audio no estaba abocado a la investigación académica y al desarrollo tecnológico, este colaboraba y participaba en diversas actividades de otros Talleres del Cmm[9]. Tales actividades incluían una relación con el Taller de Investigación y Documentación y la organización del Simposio Internacional de Música y Código /*vivo*/. Esto hace suponer que había una postura comprometida con el discurso expuesto en la descripción y objetivos del sitio web del Cmm que en parte propone una “dinámica de investigación” (CMM, s.f.) y “el desarrollo de dinámicas de experimentación que generen nuevos usos y herramientas” (CMM, s.f.) para los creadores. En ese sentido, la perspectiva del Taller de Audio más bien se apegaba al “objetivo de impulsar el uso de las nuevas tecnologías para la creación artística” (CMM, s.f.) y a la búsqueda de nuevos usos de tales tecnologías.

Lo anterior tiene que ver con lo que menciona Fernando Monreal acerca del papel que jugó el Cmm desde su creación. Monreal cuenta que gracias a la infraestructura tecnológica con la que contaban los diversos talleres del Cmm fue posible realizar una exploración de las artes electrónicas. Sin embargo, dice que “esta categoría no logró sistematizarse y convertirse en un modelo de práctica profesionalizada debido al lugar de ‘espacio común’ que se le otorgó al CMM en el organigrama del Cenart” (2020, p. 137). El autor se refiere a las artes electrónicas como una categoría que abarcaba, desde el año 1995 en que fue inaugurado este centro, al arte hecho con

9 En ese entonces los Talleres del Cmm, actualmente Laboratorios, eran: Audio, Imágenes en Movimiento, Gráfica Digital, Interfaces Electrónicas, Investigación y Documentación, Realidad Virtual y Sistemas Interactivos.

computadoras. Asimismo, por “espacio común” Monreal se refiere a la decisión de crear el Cmm como un espacio interdisciplinar y no tanto por espacios divididos en especialidades.

Lo anterior se va a ver reflejado más adelante, en los años 2010, en la aproximación espontánea que cada taller tenía con la programación de sus actividades que, si bien lograban acercar a los asistentes y participantes a tendencias artísticas con tecnologías actuales, se diluían ante la falta de sistematización que menciona Monreal. Tales actividades generaban conocimiento en el momento pero dejaban huecos al no documentarse o preservarse adecuadamente. Un ejemplo es la falta de documentación oficial organizada y mantenida sobre el live coding desarrollado en este centro[10].

De esta manera, la indagación que hacíamos los integrantes del Taller de Audio sobre el software que empleábamos y enseñábamos se inclinaba a entender su funcionamiento y aplicación en el arte computacional y no tanto a reflexionar sobre este o a impulsar un desarrollo tecnológico. El conocimiento generado era compartido en cursos, asesorías y durante las Sesiones de Live Coding. Aunado a lo anterior, había un enfoque en la difusión del software libre y de código abierto, así como en la libre distribución de los resultados artísticos y el código fuente escrito durante las Sesiones de Live Coding. Al respecto, el encuadre sobre el software libre supone una invitación a modificar los programas informáticos adscritos a esta filosofía para mejorarlos o para saber cómo funcionan. Esto supone una potencial contribución al desarrollo de lenguajes de programación para el arte o incluso el diseño de software para el mismo fin, lo que tampoco era alcanzado.

Todo lo antes expuesto coloca al contexto inicial del live coding en México como punto de partida para plantear un problema que se desprende de la relación del practicante del live coding con la tecnología que utiliza para realizar su práctica artística. El problema que encuentro es que, si bien el live coding facilita la entrada a la programación desde una perspectiva artística, generalmente lo hace desde el uso de lenguajes de programación establecidos. Tales lenguajes

10 El mayor ejemplo es el Simposio Internacional de Música y Código /*vivo*/ organizado en el año 2012 por el Taller de Audio del Cmm, que fue el tercer encuentro internacional sobre live coding realizado en el mundo. Este encuentro le dio visibilidad a México en el ámbito del live coding y provocó una serie de relaciones que hoy siguen vigentes. A pesar de la relevancia que suscitó este encuentro, más allá de la publicidad de algunos talleres y sesiones no existe información sistematizada en su sitio web. Las referencias son a través de terceros como blogs de participantes, galerías de fotos en la comunidad Flickr, entradas en Wikipedia o la recuperación del sitio web oficial del encuentro /*vivo*/ con el sistema Wayback Machine de Internet Archive.

tienen resueltas, de manera tecnológica, sus formas particulares de aproximarse a la práctica artística, lo que se extiende a su conceptualización y termina por prescribir la práctica. Además, no solo introducen una posibilidad tecnológica, también un discurso estético y un posicionamiento político. Esto no contempla la propuesta de un deslinde de los lenguajes de programación que utilizamos para realizar nuestra práctica artística sino una participación en su desarrollo tecnológico que nos permita, como artistas, implementar nuestras ideas dentro de las estructuras de tales lenguajes. La propuesta anterior es viable debido a la apertura del código fuente. Sin embargo, esto plantea el reto de trascender una barrera entre quienes practican live coding y quienes desarrollan los lenguajes de programación.

Considero necesario abordar tal problema pues el acercamiento a la programación creativa y la apertura del código fuente del software libre nos colocan en la posibilidad de realizar nuestra práctica artística vinculada a su desarrollo tecnológico. Esta posibilidad es importante pues, además de poseer una competencia técnica para escribir software, nos permite crear narrativas tecnológicas y artísticas que parten de las formas de resolver el problema computacional de la expresión artística y la imaginación de cada individuo. Asimismo, el paso que va de la práctica artística al diseño de un mecanismo de expresión dentro de un lenguaje de programación nos permite entender mejor una tecnología y entrar a las discusiones tecnológicas, estéticas, sociales y políticas en torno a ella.

Parte de esas discusiones inician con la apertura de dos aspectos: el conocimiento para realizar la práctica artística con lenguajes de programación y el código fuente del desarrollo de esos lenguajes. Desde ahí se activa la inclusión a la creación artística con lenguajes de programación y se alienta a entrar al código fuente disponible en repositorios. Este discurso se transmite dentro de las comunidades de live coding que promueven el acceso a los movimientos globales mencionados. Por ejemplo, en el caso de México, el live coding fue introducido como una práctica de programación creativa afiliada a ciertos lenguajes de programación vinculados al software libre, al código abierto y a la cultura libre. Asimismo, desde su inicio el live coding en México ha estado adscrito a la comunidad internacional Toplap y poco después al movimiento global Algorave de los que ha adoptado sus discursos y se ha adentrado a sus discusiones. Esto, aunque ha impulsado el acceso a la programación para fines creativos de muchas personas, ha

sido a expensas de una dependencia tecnológica y conceptual que, de manera posterior a la formación de la comunidad de live coding del Cmm, ha comenzado a cuestionarse desde diferentes ámbitos artísticos, tecnológicos y académicos.

Prueba de este cuestionamiento, es la serie de desarrollos tecnológicos y trabajos de investigación que se desprendieron después del periodo inicial mencionado. A excepción del desarrollo de la librería para live coding Moonlet (2011) de Elihú Pérez que fue desarrollada durante el periodo referido, se pueden mencionar trabajos en una etapa posterior como el secuenciador CaosBox de José Carlos Hasbun (2014), la librería INSTRUMENT de Rodrigo Frenk (2017), el generador de cánones Nanc-in-a-Can de Alejandro Franco y Diego Villaseñor de Cortina (2019) y el lenguaje de programación seis8s de Luis Navarro (2020). Asimismo, diversas investigaciones de maestría y doctorado se llevan a cabo en el momento de escribir esta tesis o han sido recientemente concluidas[11]. Estas investigaciones estudian al live coding, a veces de manera transversal, desde perspectivas artísticas, tecnológicas y críticas. De lo anterior puedo mencionar los trabajos con perspectiva de género de Marianne Teixido y Libertad Figueroa, el trabajo con enfoque decolonial de Luis Navarro, las investigaciones artísticas de Emilio Ocelotl, Alejandro Franco, Jessica Rodríguez y Hernani Villaseñor, y las aproximaciones científicas desde la inteligencia artificial por Iván Paz y desde las matemáticas por Ofelia Negrete. Considero importante mencionar estos trabajos debido a la relación que sus autoras y autores han tenido con el Cmm al realizar el servicio social, colaborar, estudiar o participar en sus actividades.

Ahora, aunque la apertura del código, el conocimiento y la cultura dan acceso a la práctica artística del live coding y al desarrollo tecnológico de sus lenguajes de programación, desde la entrada se establece una barrera entre quien programa con un lenguaje de programación para crear música y quien programa para desarrollarlo. Dicha barrera, que separa formas de conocimiento, pensamiento y maneras de relacionarse con la tecnología, se cae parcialmente con la apertura del conocimiento que nos dice cómo programar en el contexto de la creación artística, pero no ocurre lo mismo con la apertura del código fuente del desarrollo de un programa. La

11 Por ejemplo, la tesis doctoral *On-the-fly Syntheziser: Programming with Rule Learning* de Iván Paz (2021) o el libro *Algoritmos Arruinados: perspectivas situadas de tecnología musical* (García y Silva Treviño, 2022) que contiene diversos capítulos sobre live coding.

promesa de lo abierto en el live coding requiere tanto de conocimientos de programación que nos permitan expresarnos artísticamente como de aquellos para entender los procesos de diseño y desarrollo de software para el arte computacional.

El acceso al desarrollo tecnológico, a través de la práctica artística, puede llevarnos a entender cómo diseñar y desarrollar objetos computacionales que se incorporen a las funciones de un lenguaje de programación establecido para agregar el pensamiento artístico que parte de nuestras ideas e imaginación. De esta manera, podemos operar sobre una abstracción que nos da la libertad de expresarnos y sumar a la diversidad de modos de hacer y de pensar en el arte computacional. Esto último apunta a interrogantes sobre la transformación que sufre la práctica artística en este proceso y cómo podemos hablar de creación artística en términos de una relación que, desde el discurso, parece más comprometida con su parte tecnológica.

*Preguntas e hipótesis*

En el caso de este trabajo, la relación entre la práctica artística y el desarrollo tecnológico que planteo se desplaza en un movimiento de vaivén entre la creación musical con lenguajes de programación y la manufactura de objetos computacionales. De manera específica, en la investigación me refiero a las partes relacionadas como *formas de programar*. Así, la práctica artística es caracterizada por la *programación exploratoria* (Di Próspero, 2015, 2017, 2019) que sucede durante una presentación, concierto o ensayo de live coding con la intención de crear música en el momento; por su parte, el desarrollo tecnológico es referido como la *programación prescriptiva* –término construido con base en el término *tecnologías prescriptivas* de Ursula Franklin (1990)– que consiste en el diseño y manufactura de objetos computacionales para hacer live coding como pueden ser lenguajes de programación, librerías, *plug-ins*, clases y otras aplicaciones. La tesis establece una distinción entre estas dos formas de programar ya que en el campo del live coding tanto artistas como desarrolladores de software programan, muchas veces con el mismo lenguaje de programación, solo que con diferentes intenciones.

Asimismo, durante este trabajo he estudiado y ejercido la práctica artística de forma vinculada al desarrollo tecnológico de sus objetos computacionales bajo la siguiente propuesta:

el objeto computacional que resulta del desarrollo tecnológico forma parte de la práctica artística y dicha relación puede informar y ser informada en el contexto de la práctica investigativa. En este sentido, la finalidad del desarrollo tecnológico no es llegar a un objeto terminado autónomo sino a un objeto que actúa y se transforma junto a la práctica artística en el marco de la investigación.

De lo antes expuesto surgieron las siguientes preguntas: ¿cómo se transforma la práctica artística del live coding en el desplazamiento de la programación exploratoria a la programación prescriptiva? ¿es suficiente señalar estas formas de programación para desdibujar la supuesta barrera que existe entre la práctica artística y el desarrollo tecnológico en el ámbito del live coding? ¿cómo se relacionan ambas partes dentro de una investigación artística?

Partiendo de las preguntas anteriores, a lo largo de la tesis intentaré mostrar que la práctica artística del live coding sufre una transformación en la medida que accede a diversas capas del desarrollo tecnológico que están vinculadas a los niveles de abstracción de los lenguajes de programación. En este contexto, la apertura del código fuente es la vía que permite un desplazamiento de ida y vuelta entre la práctica artística y el desarrollo tecnológico, lo que se observa de manera concreta entre las formas de programar exploratoria y prescriptiva del live coding. Esto, para la música, significa un marco de entendimiento de los procesos que rigen la creación musical con lenguajes de programación en la práctica del live coding, los cuales parten de la implementación de las ideas artísticas en el desarrollo de objetos computacionales.

Para mostrar lo anterior diseñé un proyecto de live coding llamado SonoTexto que consta del desarrollo de tres objetos computacionales (clases de SuperCollider) y la práctica artística realizada con esos objetos. Este proyecto me permitió plantear la relación entre la práctica artística caracterizada por la *programación exploratoria* y el desarrollo tecnológico por la *programación prescriptiva*. La forma de relacionar ambas partes fue a través de la *transposición* (Braidotti en Schwab, 2018) de sus conocimientos que permite la *apertura* del código fuente y que, en este caso, apliqué a la característica de *abstracción* de los lenguajes de programación. Todo esto bajo una perspectiva de *investigación artística en tecnología musical* y situado en el contexto del live coding en México con referencia a una línea histórica que conecta al live

coding con el arte y la música computacionales. El portafolio de este proyecto puede consultarse en el siguiente sitio web: https://hernanivillasenor.com/html/sonotexto.html.

Mi intención por mostrar lo anterior ciertamente surgió de la inquietud por realizar mi práctica artística con software personalizado, a diferencia de usar la versión *vanilla* del programa[12]. Sin embargo, no se redujo a la pregunta de cómo desarrollar software sino que se convirtió en parte de la metodología de una investigación artística. Esto me permitió reflexionar sobre los discursos alrededor de la apertura de los lenguajes de programación para realizar live coding que, contrario a motivar una expresión artística personal, generan dependencias tecnológicas y discursivas. Por ello, me pareció necesario incorporar el desarrollo tecnológico como vehículo de reflexión. Esta decisión involucra una perspectiva distinta al desarrollo y publicación de un software como resultado de una investigación de tecnología musical, lo que inhibe la reflexión crítica sobre las causas, intereses y motivaciones para intercambiar conocimientos entre la creación artística y el desarrollo de tecnológico.

*Objetivos*

Los objetivos de este trabajo fueron: 1) estudiar la relación entre práctica artística y desarrollo tecnológico en el live coding a través de sus modos de programar, 2) desarrollar un objeto computacional informado por la práctica musical y la problemática de la investigación para realizar live coding, 3) estudiar la forma de implementar un modo de expresión musical en el lenguaje de programación desde la práctica artística a través del elemento *clase* de SuperCollider, 4) reflexionar sobre la música que surge de este proceso, 5) ubicar al live coding dentro de una línea histórica referente al arte computacional y la música por computadora y 6) visualizar una oportunidad de estudio del live coding dentro del campo de la tecnología musical desde la perspectiva de la investigación artística.

12 Una versión *vanilla* es definida por el *Jargon file* como una versión estándar u ordinaria de un software o un hardware (Raymond, 2004).

*Sobre el marco teórico conceptual*

El propósito de la investigación presentado en las secciones anteriores se acercó al problema de la investigación a partir de la construcción de un soporte teórico, histórico, artístico y tecnológico. Con este fin, he tomado en consideración lecturas de diferentes disciplinas como la tecnología musical, las ciencias de la computación, los estudios del software, la antropología y la tecnología.

De la revisión bibliográfica sobresalen los artículos escritos por Carolina Di Próspero de quien tomo el término *programación exploratoria* (2015, 2017 y 2019) y el libro *The Real World of Technology* de Ursula Franklin (1990) de quien tomo los términos *tecnología holística* y *tecnología prescriptiva*, el primero lo uso para complementar el término de programación exploratoria de Di Próspero y el segundo para adaptar lo que llamo *programación prescriptiva*. Asimismo, el capítulo *Transformations* de Rosi Braidotti publicado en 2006 y revisado para su inclusión en el libro *Transpositions: Aesthetico-Epistemic Operators in Artitistic Research* (2018) me sirvió para definir el desplazamiento del conocimiento entre formas de programación. Para abordar el tema del código fuente en el ámbito de la tecnología musical tomé los libros *The philosophy of software* de David Berry (2011) y *Sonic Writing* de Thor Magnusson (2019). El estado del arte del live coding lo entiendo desde las memorias de los congresos International Conference on Live Coding (ICLC) y New Interfaces for Musical Expressions (NIME), así como la revista en línea *Computer Music Journal*. De estos artículos recurro en particular a *Embodiment of code* de Marije Baalman (2015) de quien tomo el término y lo traduzco como *in-corporación del código*. También me apoyé en libros de las ciencias de la computación, principalmente, *The Good Research Code Handbook* de Patrick J Mineault (2021), *Fundamentos de programación: Algoritmos, estructuras de datos y algoritmos* de Luis Joyanes Aguilar (2020) y en el ámbito de los estudios del software recurrí en especial a los libros *Critical Code Studies* de Mark Marino (2020) y *Aesthetic Programming: A Handbook of Software Studies* de Winnie Soon y Geoff Cox (2020).

La revisión bibliográfica me ayudó a fundamentar el sistema donde se mueve el código fuente dentro del live coding artístico, así como las características de abstracción y apertura aplicadas, teórica y tecnológicamente, a la programación de la clase/objeto SonoTexto. A partir

de lo anterior, llegué a los términos que refieren las partes de mi estudio: *programación exploratoria*, *programación prescriptiva*, *apertura*, *abstracción y transposición*. Asimismo, la revisión bibliográfica me permitió observar que, en general, la investigación del live coding está orientada al desarrollo de software en los campos de las ciencias de la computación y la ingeniería de software y a una diversidad de temas de diferentes campos que no sugieren una metodología de estudio propia. Esto sigue colocando al live coding como una actividad estudiada en mayor proporción desde ámbitos científicos y tecnológicos y en mucho menor proporción desde ámbitos artísticos y humanísticos. Es aquí, precisamente, donde radica el aporte principal de la presente investigación.

*Acerca de la metodología*

Este trabajo se sustenta en dos perspectivas metodológicas: la investigación artística y las humanidades digitales. La primera negocia sus preguntas y objetos de estudio desde, y para, la práctica y la creación artística (López-Cano y San Cristóbal, 2014). La segunda incorpora metodologías de las ciencias de la computación a sus prácticas investigativas (Hayles, 2012). Ambas corrientes trataron de dirigir el estudio del live coding en esta investigación hacia una perspectiva de *investigación artística en tecnología musical*.

El término anterior no es una convención en sí, más bien una aproximación inspirada por diferentes enfoques y propuestas de investigación como *Artistic Research in Music* de Orpheus Instituut y *Artistic Research and Development* de NTNU, que incluyen a la tecnología en su eje central. Asimismo, los casos de los grupos de investigación *Music, Thought and Technology* (MTT) coordinado por Jonathan Impett y Juan Parra Cancino en el Orpheus Instituut o el *Music Technology Group* de la Universidad Pompeu Fabra. También el enfoque de la licenciatura *Música y Tecnología Artística* de la ENES Morelia, el libro *Explorations in art and technology* de Candy y Edmons (2018) que describe algunos procedimientos artísticos y, finalmente, por planteamientos metodológicos basados en la práctica artística y musical como los que proponen Henk Borgdorff (2012), Rubén López-Cano y Úrsula San Cristóbal (2014) y John Young (2015).

La perspectiva metodológica propuesta tiene que ver con dos debates de la investigación en artes y humanidades. Por un lado, se refiere al encuentro de las prácticas artísticas con la investigación académica en el proceso de conversión de las escuelas de artes en facultades iniciado en los años 1990. Este cambio trae consigo el debate sobre las formas de producir, evaluar y publicar conocimiento desde la práctica artística (Sormani et al., 2019). Por otro lado, se refiere al giro que dan las humanidades hacia la computación, el cual ha transformado la forma de investigar, leer, escribir y publicar textos científicos debido al cambio del medio impreso al digital y a la aparición del Internet y los medios digitales (Hayles, 2012).

A partir de lo anterior, las preguntas de investigación fueron generadas desde la práctica musical del live coding que incluye al desarrollo tecnológico de su objeto computacional. Para estudiar la relación propuesta a través de las formas de programación mencionadas, programé un objeto computacional que consiste en un conjunto de *clases* escritas con el lenguaje de programación SuperCollider y realicé la práctica artística como una serie de presentaciones de live coding con dichas clases. Lo anterior lo llevé a cabo bajo la idea de SonoTexto, una clase cuya función es capturar fragmentos de sonido del espacio del concierto en el momento mismo de la presentación de live coding para manipularlos con código fuente. Este término hace referencia al sonido de un lugar y al texto computacional que supone el código fuente escrito durante las presentaciones de live coding.

Con base en lo anterior, delimité la *unidad de análisis* (Güerca Torres et al., 2016) a los elementos de la problemática que enmarcan al proyecto *SonoTexto*: la abstracción del lenguaje de programación, la apertura del código fuente, la práctica artística y el desarrollo tecnológico. Lo anterior se observa en una gráfica que muestra el espacio delimitado por el código fuente abierto escrito por un humano en un espectro de abstracción alto y bajo que está vinculado a la programación exploratoria de la práctica artística y a la programación prescriptiva del desarrollo tecnológico (véase *Figura 3.2*).

La propuesta de pasar de la programación exploratoria a la programación prescriptiva se concreta en el paso de *entender qué hace* el software a *saber cómo hacerlo*. Para ello, partí de la

filosofía del software libre que propone cuatro libertades[13] de las que la denominada *libertad 1* plantea estudiar el código fuente de un programa para eventualmente modificarlo (GNU, 2021). Esta cláusula fue el eslabón que conectó la práctica artística con el desarrollo tecnológico. Si bien la apertura y la compartición de conocimiento en el live coding potencian el acceso al desarrollo tecnológico, esto no es automático; la invitación a entrar está hecha pero hay que provocarla. Con ello no me refiero a pasar de un punto a otro sino de transitar entre ambas partes con un movimiento de ida y vuelta.

En una primera fase, reflexioné sobre las circunstancias en las que hago live coding para poder entender qué objeto computacional necesitaba programar. Así, llegué a la idea de programar una clase para grabar sonido en el momento de una presentación. Esta tarea sirvió para estudiar cómo están hechas y organizadas las clases en un lenguaje de programación orientado a objetos. Asimismo, fue la pauta para reflexionar sobre conceptos de las ciencias de computación como la *abstracción*, el *lenguaje de programación*, el *código fuente*, la *apertura* y su aplicación en la práctica artística.

La segunda fase consistió en realizar presentaciones con las clases SonoTexto. Esto fue con la finalidad de poner a prueba el objeto computacional e informar a la investigación. El planteamiento supuso llevar a cabo diversas presentaciones de live coding que ocurrieron en dos etapas. La primera sucedió en espacios públicos con audiencia entre el inicio de la investigación en 2018 y la mitad de esta en 2020. La segunda se llevó a cabo en línea a partir de marzo de 2020 debido al inicio de la pandemia de COVID-19. A pesar de que esta situación modificó la forma de presentar la práctica artística, al desplazarse del espacio público al espacio privado, el problema de investigación mantuvo su perspectiva de estudio.

La tercera fase requirió la observación de la práctica artística de las presentaciones en vivo y en línea para lo que empleé el cuaderno y el registro de campo, técnica descrita por López Cano y San Cristóbal Opazo (2014) como estrategias de la etnografía aplicadas a la investigación artística. La observación de la práctica artística fue durante presentaciones en vivo, presentaciones en línea y de registros de video en canales de YouTube y Archive. La observación

13 La filosofía del software libre expresada por el proyecto GNU menciona cuatro libertades del usuario: “(0) ejecutar el programa, (1) estudiar y modificar el código fuente del programa, (2) redistribuir copias exactas y (3) distribuir versiones modificadas.” (GNU, 2021).

y análisis no se limitó solo a mi práctica artística sino a la que hacen live coders, que, en algunos casos, también desarrollan el software.

La aproximación metodológica empleada presentó la dificultad de construir el estudio de caso al mismo tiempo que era estudiado y, además, quien lo investigaba era también quien lo diseñaba y lo practicaba. Dicha dificultad me llevó a elegir una perspectiva de investigación artística para resolver la problemática planteada. Sin embargo, no fue suficiente elegir tal perspectiva para afrontar el estudio de caso, ello requirió definir cómo se lleva a cabo una investigación de este tipo en el campo de la tecnología musical, lo que representó un reto de carácter epistémico que se sumó a la problemática de la investigación. Los siguientes capítulos muestran los aspectos metodológicos, teóricos, históricos, tecnológicos y artísticos que sustentan al proyecto de live coding SonoTexto como caso para estudiar la transposición entre la práctica artística y el desarrollo tecnológico.

*Organización de los capítulos*

La tesis está dividida en cinco capítulos. El primero, *Investigación artística en tecnología musical*, discute la relación entre investigación, desarrollo tecnológico y práctica artística. Esta sección sitúa la investigación del live coding musical desde la perspectiva de las humanidades digitales, la tecnología musical y la investigación artística. Asimismo, expone el papel del código fuente en una investigación de este tipo y, de manera particular, los términos *exploración*, *prescripción*, *apertura* y *transposición*. El capítulo tiene la función de presentar el terreno metodológico de la tesis y las nociones que delimitan la unidad de análisis que será desarrollada durante el tercer capítulo.

El segundo capítulo, *Live coding: arte y música computacionales*, traza una línea histórica entre el arte computacional y el live coding, la cual atraviesa por la música por computadora. A partir de esto, expone al live coding según el manifiesto escrito por el colectivo Toplap en el año 2004. A partir de ello, el capítulo discute cinco ejemplos de desarrollos tecnológicos dedicados a esta práctica y expone una perspectiva del estudio del live coding. Esta sección busca colocar el live coding en un contexto cultural más amplio y mostrar el vínculo

entre la práctica artística y el desarrollo tecnológico en los ejemplos específicos presentados. De manera particular muestra el contexto, el campo de estudio y la perspectiva de investigación de este trabajo.

El capítulo tres, denominado *Apertura y abstracción en la práctica del live coding*, trata sobre los niveles de abstracción de los lenguajes de programación y la apertura del código fuente en los programas. Los términos *alto, bajo, abierto* se refieren al vínculo entre los niveles de abstracción en la programación, la práctica artística y el desarrollo tecnológico en un contexto de apertura de código y conocimiento. Esta sección reconoce las partes involucradas en la transposición de conocimientos entre la programación exploratoria y prescriptiva para analizar la relación entre la práctica artística y el desarrollo tecnológico en el caso del proyecto de live coding SonoTexto.

El capítulo cuatro se denomina *Desarrollo tecnológico, investigación y práctica artística en el proyecto SonoTexto*, y presenta el objeto computacional y la práctica artística realizadas durante la investigación. En este capítulo cuento cómo surge la idea de SonoTexto; describo los aspectos relacionados al objeto computacional, su desarrollo y funcionamiento; asimismo, expongo los aspectos relacionados con la práctica artística, derivados de las presentaciones de live coding realizadas con el objeto computacional y finalmente discuto cómo se retroalimentan las partes involucradas. Además de la descripción de los procesos tecnológicos y artísticos, el capítulo vincula las nociones expuestas en los capítulos anteriores al desarrollo tecnológico y a la práctica artística llevadas a cabo con el proyecto SonoTexto.

Por último, el capítulo cinco que lleva por título *SonoTexto: investigación orientada a la transposición entre desarrollo tecnológico y práctica artística* está dividido en cuatro secciones que discuten los alcances de la tesis. Primero expone cómo la apertura, la abstracción y la transposición actúan en la programación exploratoria y en la programación prescriptiva; después se ofrece una discusión de las categorías de la tesis agrupadas en modos de programar y la perspectiva de la investigación de la tesis.

# 1 Investigación artística en tecnología musical

En el contexto de la universidad, el campo de la tecnología musical está vinculado a los ámbitos de la investigación, la práctica artística y el desarrollo tecnológico. Estos ejes se mezclan de manera particular bajo un anhelo por lo multi e interdisciplinario debido a los diferentes campos que atraviesan el área de tecnología musical como la música, la física, las matemáticas, las ciencias de la computación y las humanidades. No obstante, es un área fragmentada por la especialización disciplinar, en la que los campos mencionados, pertenecientes al arte, la ciencia o la tecnología, pueden conectarse desde la perspectiva que cada quien decida implementar en su proyecto de investigación. En el caso particular de este trabajo, enmarcado por el campo de la tecnología musical, la decisión ha sido abordarlo desde una perspectiva de investigación artística.

Lo anterior plantea el problema de la metodología a utilizar para resolver las preguntas o hipótesis planteadas, pues cada disciplina utiliza paradigmas y estrategias distintas. Aunado a esto, la falta de modelos (Borgdorff, 2012; López-Cano y San Cristóbal, 2014) en el ámbito de la investigación artística en música requiere la construcción de una metodología con base en la apropiación de estrategias y métodos de otros campos. En este caso, la indagación comenzó desde mi práctica artística de live coding, situada en un contexto específico, que buscó vincularse a su desarrollo tecnológico para dar respuesta a la problemática planteada. Esta práctica involucra la programación con lenguajes de programación dedicados a crear música *en el momento*[14] y la programación estructurada para la manufactura de una pieza de software u objeto computacional para llevar a cabo la creación musical en el live coding.

Este capítulo articula la perspectiva de investigación de mi trabajo con base en la tecnología musical vista en el contexto de las humanidades digitales, lo que discuto en la primera

14 A lo largo del trabajo me inclino por la expresión *en el momento* para definir la creación musical con lenguajes de programación en el ámbito del live coding. La prefiero sobre otras expresiones como *en vivo*, que sería una traducción literal de la palabra “live” o *en tiempo real*, que es un término informático adaptado a ciertas prácticas de composición y a la improvisación musical. El término *en el momento* lo tomo del compositor e improvisador Wade Matthews (2012) quien lo utiliza en su libro *Improvisando: La libre creación musical* para calificar la improvisación como una forma de creación musical que surge de la inmediatez, el proceso y la colectividad. Con esto sugiero que el live coding, como creación musical con lenguajes de programación *en el momento*, está conectada con la cualidad improvisatoria que le atribuyen autores como Di Próspero (2015), McLean (2011) y Rohrhuber y De Campo (2009).

sección. Después, la segunda sección trata sobre el código computacional dentro de la investigación musical. La tercera sección presenta las nociones de *programación exploratoria*, *programación prescriptiva* y *apertura* que sirven para caracterizar y vincular dos modos de escribir código fuente en el live coding que estarán presentes durante el texto. Finalmente, para elaborar estos cruces utilizo las ideas de *transposición* y *giro*, las cuales describo en la cuarta sección.

## 1.1 Investigación en humanidades digitales y tecnología musical

Esta investigación parte de una práctica artística que se expresa con la escritura de código fuente con un lenguaje de programación. Dicha práctica consiste en la programación de una computadora durante un concierto para generar sonido e imagen. Una tarea de este tipo se realiza con lenguajes de programación de sintaxis sencilla y precisa que incluye palabras del lenguaje natural humano, generalmente inglés, que permite a la persona que programa expresarse artísticamente con estructuras computacionales y musicales. La programación de computadoras, en la música y el arte, se vale de estructuras lógicas, matemáticas, musicales, visuales y lingüísticas que la hacen un objeto de estudio de diferentes disciplinas como las ciencias de la computación, la música, la imagen, la literatura o las humanidades.

Este trabajo se sitúa en el campo de la tecnología musical, un área que en el posgrado donde fue realizado es incluida en el área de las humanidades y las artes. Más allá de la categoría que ubica el campo de mi investigación en un área específica, reconozco que el campo de las ciencias de la computación es necesario para entender los procedimientos computacionales que estructuran la práctica del live coding. El hecho de contar con un área de tecnología musical parece una razón obvia para llevar a cabo esta investigación, sin embargo, lo que determina la perspectiva de este trabajo es la problemática que se encuentra entre la práctica artística digital y la manufactura de los objetos computacionales que requiere para llevarse a cabo. Lo anterior me posiciona en el estudio de objetos de las ciencias de la computación vistos desde las

humanidades. Una perspectiva así puede observarse en las humanidades digitales desde donde derivan campos de estudios que incluyen el software, el código de computadoras y la implementación de algoritmos. Es aquí donde el campo de la tecnología musical, como área en la que convergen distintas disciplinas, resulta idóneo para desarrollar este trabajo.

Un ejemplo de lo anterior es descrito por Matthew Fuller (2008) en la introducción del libro *Software Studies: a lexicon*. El autor plantea que este campo tiene una perspectiva crítica sobre el estudio de objetos digitales, lenguajes y estructuras lógicas que abarca algoritmos, formas de pensar y hacer, juicios de valor y estética de la computación, así como subculturas de programación y sus políticas. Fuller señala que el software es un objeto de estudio para diferentes áreas distintas a las que tradicionalmente abarcan el software. Estas áreas, dice el autor, se identifican con la cultura y los medios desde una perspectiva política, social y estética. Por su parte, David Berry y Michael Dieter (2015) en el capítulo de introducción *Thinking Postdigital Aesthetics: Art, Computation and Design* plantean que hay un giro computacional en nuestras vidas en el que diversos algoritmos corren continuamente en los sistemas técnicos de software insertos en nuestra cotidianidad, los cuales median, crean y representan el mundo. Ante esto, los autores señalan un tipo de investigación crítica desplazada a una forma computacional de diversas áreas de estudio entre las que se encuentran las humanidades digitales, los estudios del software y los medios computacionales.

Jordi Vallverdú y Alicia Fornes describen las humanidades digitales como un “área emergente e interdisciplinar en la que convergen las disciplinas humanísticas y sociales con las tecnologías de la información y la comunicación” (en Vallverdú, s.f.). Estos académicos explican que tal convergencia ha posibilitado un cambio en la investigación y estudio en las humanidades con el uso de métodos que emplean software para el análisis de grandes bases de datos abiertos de disciplinas como la música digital, la etnografía digital, el pensamiento hipertextual y el patrimonio digital por mencionar algunas.

Por su parte, Katherine Hayles (2012) en su libro *How we think?* narra el proceso por el que pasaron algunos departamentos de literatura y humanidades de universidades en Estados Unidos y Reino Unido para integrar sus procesos de investigación y educación a un ámbito digital. La autora menciona que la computadora y los algoritmos de análisis jugaron un papel

importante para definir el campo de las humanidades digitales, lo que implicó colaborar con otros campos de conocimiento como las ciencias de la computación. El término humanidades digitales, dice la autora, aparece en 1999 para reemplazar lo que se conocía desde los años 1940 como *humanities computing*. Más que el trabajo con bases de datos, explica, las humanidades digitales proponen nueva bibliografía que se ha movido del enfoque en la codificación de textos y su análisis a un énfasis en prácticas multimedia. En este giro las humanidades basadas en texto se juntan con prácticas temporales como el cine y la música, prácticas de tradición visual como la gráfica y el diseño, prácticas espaciales como la geografía y la arquitectura y finalmente con prácticas curatoriales. Para la autora, las humanidades digitales son "un campo diverso de prácticas asociadas con técnicas computacionales y con un alcance más allá de lo impreso en sus modos de indagar, investigar, publicar y diseminar" (Hayles, 2012, p. 27).

Por otro lado, la tecnología musical es un campo de estudio relativamente nuevo, el Posgrado en Música de la UNAM (s.f.) la define en su sitio web a partir del uso de tecnología electrónica, informática, de grabación y de reproducción sonora en la música y el sonido, así como la integración de conocimientos de las matemáticas, la acústica, la psicología, la ingeniería, las artes visuales y la estética. Esta descripción coincide con lo que dice Anna Xambó (2016), quien define la tecnología musical como un campo nuevo e interdisciplinario que toma métodos y técnicas de las ciencias, la ingeniería, las ciencias sociales y las artes. Por su parte, Andrew Brown (2007) la describe a partir de un recorrido histórico que incluye diferentes desarrollos musicales y tecnológicos como la notación musical, el descubrimiento de metales que propiciaron la construcción de nuevos instrumentos acústicos y la incorporación de tecnologías electrónicas y de grabación que permitieron el desarrollo de la música electrónica y concreta a mediados del siglo XX.

Si bien el encuentro entre las humanidades y las ciencias de la computación ha conformado el campo de las humanidades digitales, esto no necesariamente incluyen a la tecnología musical, al menos de manera explícita. Sin embargo, el cruce entre estos dos campos puede propiciar una perspectiva de estudio que observa los aspectos tecnológicos y artísticos de prácticas como el live coding. En este sentido, Jonathan Impett (2016) hace alusión al encuentro de las humanidades digitales con la investigación en tecnología musical cuando introduce al

grupo de investigación *Music, Thought and Technology* del Orpheus Instituut. Este músico e investigador menciona que “la investigación de las humanidades digitales y la creación basada en computadoras usan el mismo repertorio de herramientas; ambos son actos de imaginación musical extendidas y exploradas a través de la tecnología” (Impett, 2016).

Aunque muchas universidades han tenido acceso a las computadoras desde el inicio de su desarrollo electrónico, o las han diseñado[15], es relativamente reciente que áreas de las humanidades, las artes y las ciencias sociales realicen lo que David Berry (2011) refiere como *computational turn* o giro computacional que describe como un cambio teórico y conceptual de los fundamentos críticos de las disciplinas antes mencionadas. Tal giro, según Berry, ocurre al utilizar tecnologías informáticas como el uso de algoritmos para recolectar y analizar datos. Dicha preocupación se puede observar en el seminario permanente *Entornos y Narrativas Digitales en la Academia* SENDA organizado por el Instituto de Investigaciones Antropológicas de la UNAM, cuyas sesiones abordan temas relacionados a la tecnología informática y la programación en las humanidades digitales o la inteligencia artificial en la educación[16].

Además de lo anterior, diferentes casos en México muestran cómo algunas facultades y divisiones universitarias dedicadas a las humanidades incluyen cada vez más el estudio y la práctica de las artes digitales en sus programas de licenciatura o se encaminan a una “transformación digital en la enseñanza de las artes” (Rodríguez Peña, 2022). Por ejemplo, la licenciatura en *Composición* de la Facultad de Música y la licenciatura en *Diseño y Comunicación Visual* de la Facultad de Arte y Diseño de la UNAM, o las licenciaturas de *Música y Tecnología Artística* en la ENES Morelia, *Arte y Comunicación Digitales* en la UAM Lerma, así como las de *Arte digital* en la UAEMex o la *Licenciatura en Artes Visuales* de la UMSNH. En estos casos, la programación y el uso de tecnologías digitales son incluidas en materias relacionadas con la música y las artes digitales tales como informática musical, música

15 Glenn Brookshear (2012) menciona algunos ejemplos como la computadora electromecánica Mark I diseñada por Howard Aiken, construida por IBM y finalizada en 1944 en la Universidad de Harvard, o las computadoras desarrolladas por Atanasoff-Berry en 1941 en el Colegio Estatal de Iowa, así como la computadora ENIAC construida por John Mauchly y Persper Eckert en la Universidad de Pensilvania. Por su parte, Paul Doornbusch (2004) cuenta que la computadora CSIRAC, que considera la primera computadora en tocar música en 1950, fue trasladada a la Universidad de Melbourne en 1955.

16 http://132.247.215.189/SENDA/index.php.

electroacústica, síntesis de sonido, composición asistida por computadora, animación, programación web, así como arte, tecnologías y sistemas interactivos.

En estos contextos, la programación de computadoras otorga una especie de *literacidad* informática a quienes se inician en las artes digitales. Según Daniel Cassany (2005), el concepto de literacidad es un término reciente de corte sociocultural que "abarca todos los conocimientos y actitudes necesarios para el uso eficaz en una comunidad de los géneros escritos" (p. 1), esto para quien escribe y para quien lee. En este sentido, se puede tomar la noción de literacidad para aplicarla al entendimiento de código fuente en un esfuerzo por romper la barrera entre usuarios y programadores. Lo anterior se expresa mejor en una comunidad que escribe código fuente e incluye artistas que programan con fines artísticos y programadores que desarrollan lenguajes de programación y software para tal fin. Una comunidad que se forma, tanto de manera escolarizada como no escolarizada, en la escritura de código fuente podrá trascender esa barrera que separa papeles asignados a las personas que programan. Así lo expresa la programadora y artista digital Olivia Jack quien dice que "a veces hay una barrera entre sentirse como 'usuario' de algún software muy puntual o, '¡ay! sé programar pero dentro de este contexto muy específico', o de *saber cómo*, o de pensar: soy programadora" (Piranha Lab, 2019, 2:16:17). Lo anterior es expresado con base en cómo se definen (usuario o programador) quienes programan en el ámbito de live coding. Para ella 'ser programadora' es saber buscar lo que no sabes sobre programación, algo que implica una grado de literacidad sobre la escritura de código fuente.

Con lo anterior, los lenguajes de programación dedicados al arte digital son diseñados con la intención de hacer accesible la programación a los usuarios de tecnologías informáticas que no cuentan con una formación especializada. Esto, sin embargo, no es exclusivo del ámbito artístico, tiene antecedentes en entornos científicos, empresariales y militares. Por ejemplo, Marino (2020) menciona el caso del lenguaje de programación FLOW-MATIC desarrollado en 1958 por Grace Hopper y su equipo con el objetivo de facilitar la programación a empresarios y militares con un lenguaje similar al natural. También se puede mencionar el paradigma de programación orientado a objetos del que, Joyanes Aguilar (2020) cuenta que fue ideado en 1969 por Kristen Nygaard dentro del ámbito científico para facilitar el manejo de objetos y fenómenos de la vida real al ser representados con atributos y comportamientos computacionales.

Por otro lado, el grupo de investigación *Music, Thought and Technology* (MTT) del Orpheus Institute indaga sobre los conceptos de tecnología y ciencia en prácticas musicales actuales. Sus líneas de investigación abarcan desde el uso práctico de la tecnología en la música hasta la "creación, entendimiento, crítica, representación, pedagogía y discursos de la música" (Parra Cancino e Impett, 2020, p. 2). Bajo el estudio de la tecnología en la música, el grupo MTT investiga la relación entre trabajo musical y objeto digital. Según los autores estas partes se conectan por la materialidad, la inmaterialidad, la abstracción y los lenguajes formales presentes en las dos. La aproximación anterior resuena en lo que expresan Miriam Peña Pimentel y Francisco Barrón Tovar (2020) en la ponencia *Humanidades Digitales* presentada en el seminario SENDA, quienes definen las humanidades digitales como prácticas de investigación colaborativas y multidisciplinarias preocupadas por incluir diversas tecnologías digitales en la investigación humanística a partir de las que se genera conocimiento desde su propio medio. Al igual que Parra Cancino e Impett, Peña Pimentel y Barrón Tovar reconocen una influencia de modelos tecnológicos en el pensamiento y la práctica humanista.

Por último, el campo de la tecnología musical, al estar inserto en el área de las humanidades, se ve inmerso en enfoques de campos que abarcan desde las ciencias de la computación hasta las humanidades redefinidas en términos de lo digital. Este cruce, así como la inclusión de la programación en la investigación de tecnología musical, abren la oportunidad de realizar un estudio crítico con marcos de referencia de ambas disciplinas que se ven amalgamados en campos como los estudios del software (Fuller, 2008) y los *estudios críticos del código* (Marino, 2020). Estos enfoques presentan similitudes en sus discursos sobre la tecnología informática vista en calidad de herramienta de investigación, desarrollo y objeto de estudio. En este contexto, el código fuente es empleado para desarrollar tecnología y crear arte al mismo tiempo que es estudiado desde perspectivas tecnológicas, científicas, artísticas, estéticas, políticas y culturales (Berry, 2011; Berry y Dieter, 2015; Cox y McLean, 2013; Fuller, 2008; Hayles, 2012; Marino, 2020).

## 1.2 El código fuente en la investigación

Una serie de caracteres, números, notación matemática y palabras aparecen en la pantalla de una computadora cada vez que programamos. El sistema de escritura que conforma lo anterior es llamado comúnmente código fuente y se organiza con la sintaxis, el léxico y la semántica de un lenguaje de programación. El resultado de su escritura es la codificación de un conjunto de instrucciones, generalmente diseñadas en forma de un algoritmo, para que una computadora realice tareas como la generación de imagen o sonido. Desde que el código fuente entra a la computadora en su forma textual hasta que sale en forma de gráficos o sonido, este transita por estados de escritura, traducción y ejecución. Es decir, el recorrido del código fuente pasa de su estado textual escrito por un humano al código que ejecuta diferentes procesos dentro de la computadora para llevar a cabo las tareas programadas. De este tránsito, el código que escribe y lee quien programa no es el mismo que ejecuta la computadora: el primero se conoce como *código fuente* y el segundo como *código máquina*.

Además de las ciencias de la computación, el estudio del código fuente ha sido tomado por áreas de las humanidades para analizarlo más allá de su contexto funcional. De ello se desprenden perspectivas tecnológicas, humanistas, culturales, sociales, políticas y estéticas (Berry, 2011; Chun, 2008; Cox y McLean, 2013; Cramer, 2005; Marino, 2020; Soon y Cox, 2020), así como categorías de análisis entre las que se encuentran el *código crítico* (Berry, 2011; Marino, 2020), del que hablaré más adelante.

En este sentido, los diferentes términos, estados y descripciones del código dentro de una computadora son definidos, por un lado, desde las ciencias de la computación y, por otro lado, desde diferentes campos de las humanidades que lo adoptan como objeto de estudio. De las formas y categorías señaladas me enfoco en el código fuente como texto escrito por practicantes live coding y desarrolladores de software. Con ello señalo el enfoque de mi perspectiva de investigación centrado en el aspecto humano del código fuente como vehículo de expresión, comunicación y socialización en el contexto de la creación musical con lenguajes de programación.

Florian Cramer (2005) menciona que el significado computacional del término tiene que ver con “una regla que transforma símbolos en acción” (p. 9). El autor menciona que el ámbito de la programación hace referencia al código de computadoras como secuencias de instrucciones ejecutables o instrucciones algorítmicas. De manera general, el autor diferencia dos tipos de código: el código escrito y leído por humanos que recibe el nombre de *código fuente* y el código traducido para que la computadora ejecute sus tareas al que llama *código binario compilado* también referido como *código máquina*. Por su parte, Friedrich Kittler (en Fuller, 2008) hace un recorrido histórico que traza los orígenes del término *código* en la escritura secreta o encriptación, técnica empleada desde el Imperio Romano hasta la Segunda Guerra Mundial. Tal recorrido le permite argumentar a Kittler que el código no es algo particular de la tecnología computacional sino que más bien es una *tecnología de comunicación.*

Pasando al ámbito informático, el código fuente es una representación textual de lo que la computadora ejecutará cuando esta lo traduzca a su lenguaje de máquina. El código fuente muestra lo que será un proceso temporal de cómputo de la práctica artística o del desarrollo tecnológico. En cuanto al ámbito investigativo, el código fuente se incorpora al problema de la investigación como objeto de estudio que puede ser analizado como texto o que realiza acciones a través de su traducción y ejecución. Pitchard, Snodgrass y Tizlik-Carver, en la introducción del libro *Executing Practices* (2018), mencionan que el problema de la ejecución de procesos computacionales va más allá de las instrucciones que el CPU de una computadora realiza, lo que describe como una ejecución técnica y cultural. Esto abarca prácticas investigativas referentes a lo que llaman *prácticas de ejecución*, que estudian de manera crítica las implicaciones de la ejecución de instrucciones computacionales a través de una especie de lupa que aletarga el tiempo de los complejos y veloces procesos computacionales que lleva a cabo el procesador de computadora para estudiarlos desde varios enfoques.

En este punto cabe señalar que la función del código fuente como elemento que implementa instrucciones para que una computadora realice acciones está incompleta sin la parte que las conceptualiza: el algoritmo. Según Joyanes Aguilar (2020), un algoritmo, en el ámbito de las ciencias de la computación, es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos. Los algoritmos, dice el autor, son independientes de la

computadora y describen tres partes: una entrada, el proceso y una salida. Por su parte, Gerhard Nierhaus (2009) en el libro *Algorithmic Composition: Paradigms of Automated Music Generation* menciona que los algoritmos son métodos formalizados que contienen reglas para generar estructuras musicales. Distingue que en la música la idea de finitud del proceso algorítmico queda en suspenso pues en la composición algorítmica algunas salidas no son definitivas ya que involucran decisiones de probabilidad en la solución del problema. Para Nick Collins (en Dean y McLean, 2018), la composición algorítmica es una forma de pensar la música que se extiende a través de la historia. Señala que el pensamiento algorítmico no está necesariamente asociado a la computadora. Antes que procedimientos para generar música ve en ello una conexión entre música y matemáticas que se extiende desde la investigación pitagórica de las relaciones numéricas y musicales, el quadrivium medieval, la combinatoria de las matemáticas discretas hasta la música por computadora actual.

Pero, ¿qué relación tienen los algoritmos con el código fuente? Según Joyanes Aguilar (2020), antes de escribir un programa es necesario el diseño de sus algoritmos cuyas soluciones instrumentalizadas por las serie de instrucciones son implementadas como código fuente al elaborar un programa. De esta manera, la escritura de un algoritmo que conforma un programa se realiza con un lenguaje de programación, cuya actividad se conoce como codificación o programación. Así, los programas informáticos se componen por código fuente que no son otra cosa que algoritmos escritos. En este sentido, el código fuente es la vía que implementa las instrucciones de un algoritmo. Es el portador de la solución descrita en el diseño de un algoritmo que resuelve un problema de cómputo. Así, para que un algoritmo sea ejecutado por una computadora requiere ser codificado con un lenguaje de programación. Lo que resulta de este proceso es el código fuente. Cabe aclarar que en el presente trabajo me inclino por la noción de código fuente como texto tecnológico y cultural que enuncia acciones y no en la noción de algoritmo, aunque estoy consciente de que el código fuente es una implementación refinada del algoritmo.

Aclarado lo anterior, el interés por la programación de computadoras en el ámbito de la investigación musical se manifiesta en propuestas de carácter creativo y formal. Por un lado, están las prácticas artísticas realizadas con un lenguaje de programación; por otro, los desarrollos

de objetos computacionales diseñados para llevar a cabo una tarea musical. Cada una de estas aproximaciones requiere saber programar. Sin embargo, para realizar una investigación no basta conocer cómo funciona un lenguaje de programación para la creación artística o los procedimientos del desarrollo tecnológico, es necesario cuestionar la actividad de programar más allá de lo funcional. Esto implica entender la carga cultural plasmada en el software y en la escritura de código fuente para realizar ambas tareas. Lo anterior remite a lo que comenta Fuller (2008) sobre los estudios del software, los cuales nos permitirían ver el software desde diversas formas de existir, experimentar y pensar; asimismo, a su planteamiento acerca de que el software es un objeto de estudio para disciplinas que están comprometidas con la cultura y los medios desde posturas políticas, sociales y estéticas.

Para llevar a cabo estas tareas, el código fuente sirve como elemento estructural en la creación y desarrollo de interfaces, instrumentos digitales, composiciones, improvisaciones, diseño de algoritmos, sistemas de inteligencia artificial, aprendizaje y escucha de máquinas, música algorítmica, música generativa y live coding. La lectura y escritura del código fuente se integran al campo de la música y de las humanidades desde la perspectiva de la creación artística, el desarrollo tecnológico y la investigación. Esto supone en el contexto de la investigación una aproximación para entender qué es y qué hace el código fuente.

Desde la perspectiva de esta investigación, leer y escribir código fuente son dos tareas distintas. Leer código se refiere al análisis del código escrito por un humano para dar instrucciones a una computadora. La lectura de código fuente, además de analizar sus funciones, se enfoca en el contexto cultural en el que fue escrito, perspectiva que viene de los estudios críticos del código (Marino, 2020). Por otra parte, escribir código fuente se refiere a la organización del conjunto de palabras, símbolos y caracteres que conforman a los lenguaje de programación con el objetivo de programar las tareas que realiza una computadora. Ambos planteamientos son importantes pues nos dan herramientas para hacer software y reflexionar sobre él, en palabras de Soon y Cox “leer, escribir y pensar con software” (2020, p. 13).

Un tema que discute Hayles (2012) es la necesidad de que académicos de las humanidades puedan leer, interpretar y escribir código computacional para incorporarlo a sus metodologías de investigación. Por su parte, Marino (2020) plantea desde las humanidades un

acercamiento distinto con los *estudios críticos del código*. El autor define esta área como un campo de estudio multidisciplinario que indaga en el significado del código, no solo en lo que hace funcionalmente, sino desde lecturas hechas en distintas temporalidades y contextos tecnosociales. Marino plantea una metodología de lectura del código fuente que indaga dentro de su contenido y contexto. Esta aproximación propone el estudio y el análisis del código fuente como un texto desplazado de su ejecución computacional con la intención de comentarlo, documentarlo e interpretarlo de manera hermenéutica desde la teoría crítica y en colaboración con las ciencias de la computación. En el ámbito de las nuevas tecnologías para la expresión musical (NIME), Magnusson (2019) plantea la necesidad de aprender a leer código computacional para analizar obras musicales creadas con lenguajes de programación en una computadora. Por su parte, Maxence Larrieu (2019) añade que en el campo de la música por computadora no se ha hecho mucho énfasis en el código fuente cuando se analizan piezas de música computacional. Según el autor, esto se debe a que la música por computadora ha sido incluida históricamente en el espectro de la música electroacústica que se enfoca en la señal de audio como medio de creación y no en los sistemas de notación que supone su uso en la creación musical con lenguajes de programación.

De los señalamientos anteriores, Marino (2020) es quien va más lejos en el estudio del código fuente. El autor propone en su libro sobre estudios críticos del código una metodología de estudio de caso de diferentes proyectos, de los que imprime el código fuente en el libro. A partir de ello explica la funcionalidad de algunas líneas y discute ciertos comentarios escritos dentro del archivo original del código fuente. Con estos elementos analiza y discute su significado fuera del contexto de ejecución en una computadora.

Por ejemplo, uno de los casos que analiza Marino es la herramienta de supervivencia basada en datos de geolocalización *Transborder Immigrant Tool* (TBT) creada y desarrollada en el 2007 por Electronic Disturbance Theater (EDT), grupo residente en la Universidad de San Diego California formado por académicos, académicas, programadores y artistas. El proyecto consiste en una aplicación de teléfono móvil dirigida al migrante que cruza la frontera entre México y Estados Unidos a través del desierto. Tal aplicación indica algunos puntos donde se han colocado bidones de agua y reproduce archivos de audio con consejos de supervivencia en

forma de poesía. El código fuente analizado por Marino, que consta de 597 líneas, es colocado al inicio del capítulo dedicado a este proyecto junto a una ficha técnica que incluye la autoría, librerías y lenguajes de programación utilizados, así como la fecha de desarrollo, entre otros datos. El autor realiza un amplio análisis del proyecto que incluye aspectos técnicos, artísticos y políticos, de los que se enfoca mayormente en su carácter activista y poético. A partir de una metodología comparativa propia de la literatura, el autor desarrolla un argumento que vincula la estructura de la poesía del proyecto, basada en procedimientos o instrucciones de supervivencia, con el código fuente de la aplicación.

Por otro lado, Geoff Cox realiza un análisis parecido en el libro *Speaking Code* (2013). Con la ayuda del programador y live coder Alex McLean, colocan fragmentos de código en el libro para ejemplificar su argumento acerca de la legibilidad del código. Para Cox y McLean existe una relación entre texto y código fuente que puede conjuntarse desde la escritura crítica de las humanidades y el desarrollo de software, bajo la premisa de que “el habla dice lo que hará y lo hace” (p. xiii). Cox y McLean (2013) mencionan que el código es escritura, interpretable, notación legible para la computadora y representación de su proceso como si fueran palabras dichas.

En los casos anteriores el código fuente se convierte en objeto de estudio y es parte de los materiales de una investigación. La práctica artística, el desarrollo tecnológico y la investigación tienen en el código fuente un nodo de convergencia que, desde el live coding, permite construir la perspectiva del presente trabajo. Esto presenta el reto de colocar el código fuente junto al lenguaje verbal escrito. Pero, ¿qué dice el código fuente impreso en una tesis o un libro de humanidades y artes? Incluir piezas de código fuente es una práctica común en los libros de programación, sin embargo, lo que distingue los ejemplos mencionados es que se encuentran en el ámbito de las humanidades desde donde se analiza el código computacional con una perspectiva distinta a las ciencias de la computación. Sin embargo, este trabajo no pierde de vista la parte operacional del código fuente pues, como se mostrará en los capítulos 3 y 4, el funcionamiento del código fuente aplicado al proyecto SonoTexto es parte de lo que muestra la apertura. En todo caso, ambas perspectivas son importantes para abordar el problema y no

enfocarlo en una cuestión de desarrollo tecnológico *per se* o solo en el análisis desde el ámbito de las humanidades.

## 1.3 Exploración, prescripción y apertura

En esta tesis discuto dos tipos de conocimiento acerca de la programación de computadoras en el ámbito del live coding: 1) el conocimiento para crear música con un lenguaje de programación que responde a la pregunta *qué hace* tal lenguaje de programación y 2) el conocimiento para diseñar e implementar el lenguaje de programación que responde a la pregunta *cómo hacer* (parte de) ese lenguaje de programación. Dichos conocimientos los he caracterizado como formas de programar que establecen dos categorías de la tesis. Una es la *programación exploratoria* u *holística* y la otra es la *programación prescriptiva*. Estas categorías las construí a partir de las ideas y conceptos de Carolina Di Próspero (2015, 2017 y 2019) y Ursula Franklin (1990).

*Programación exploratoria*

Primero voy a exponer las ideas detrás de la categoría de programación exploratoria que tomo del trabajo de la antropóloga social Carolina Di Próspero. Esta autora hace una constante referencia al carácter e intención exploratorios en el trabajo etnográfico sobre live coding que expone en el artículo *Live coding. Arte computacional en proceso* escrito en el año 2015. El texto trata sobre la práctica artística del live coding musical basado en técnicas de observación y en entrevistas que la autora realizó con diversos live coders. A partir de esta investigación, Di Próspero publicó dos artículos subsecuentes en los que vincula al live coding con un tipo de código artesanal, con la apertura y con el proceso. En el artículo publicado en 2017, Di Próspero observa en sus discursos una intención de “superar y dejar atrás es [sic] cierta perspectiva convencional de la programación, a partir de explorar artísticamente, construyendo nuevas relaciones con la tecnología” (2017, p. 5).

Di Próspero menciona que en el caso del live coding la palabra programar supone una *intención de explorar* durante una *performance de live coding*. Esta forma de programar tiene el objetivo de improvisar música mediante la escritura de los símbolos y las palabras que forman la sintaxis de un lenguaje de programación determinado. El proceso ocurre al escribir y modificar código fuente con un lenguaje de programación dedicado a producir sonido o imagen con la intención de expresar ideas artísticas. Esta aproximación tiende a ser un tanto arbitraria en el empleo de los recursos de la programación y se aproxima mediante la prueba y el error de los mismos. Al respecto, la investigadora y compositora computacional Shelly Knotts expresa lo siguiente: “el live coding se trata de explorar el fallo y el error” (Knotts y Armitage, 2015), lo que complementa la también investigadora y compositora Joanne Armitage “hay una verdadera inmediatez en ello, es probar cosas inmediatamente y ver si funcionan o no” (Knotts y Armitage, 2015).

A partir de las ideas y conceptos anteriores, el término *programación exploratoria* es una forma de escribir código fuente que sucede en la práctica de live coding con una intención artística en la que se toman decisiones en el momento basadas en la experiencia de programación. Este término se complementa con el concepto de tecnología holística de Franklin que refiere un proceso artesanal y un saber hacer único de cada persona.

### *Programación prescriptiva*

En cuanto a la categoría de programación prescriptiva, expongo a continuación los conceptos e ideas desde donde la formulé. Ursula Fanklin, en el libro *The Real World of Techology* (1990), distingue el desarrollo tecnológico en dos formas: las *tecnologías holísticas* y las *tecnologías prescriptivas*. La autora expone que “[l]as tecnologías holísticas están normalmente asociadas con la noción de artesanía” (p. 18), en las que el artesano “controla el proceso de su trabajo de principio a fin” (p. 18). Franklin argumenta que los artesanos toman decisiones en el momento de su trabajo que aplican a situaciones únicas. Opuesto a lo anterior, dice que hay una “especialización por proceso; la cual llam[a] tecnología prescriptiva. […] Aquí, el fabricar o hacer algo se descompone en pasos claramente identificables” (p. 20). Cada paso, continúa, lo

realiza un trabajador distinto, quien está familiarizado solo con ese proceso, lo que es llamado división del trabajo.

La noción de división de tareas se confirma en lo que expresa el ingeniero en informática Luis Joyanes Aguilar (2020), quien cuenta que en el mundo de la programación esta idea es una cualidad del proceso de desarrollo de un programa que apunta a dividir los problemas complejos en pequeños subproblemas para facilitar su solución.

De esta manera, el término *programación prescriptiva* es una adaptación que hago del concepto de tecnología prescriptiva de Franklin, el cual apunta a una programación especializada en el ámbito de la informática que requiere seguir pasos y protocolos determinados para desarrollar, en este caso, objetos computacionales. De esta manera, la programación sigue un proceso que involucra los pasos de diseño, escritura e implementación del código fuente que nos llevan a solucionar un problema computacional que plantea la realización de una tarea de cómputo. Tales pasos son enumerados por Joyanes Aguilar (2020) en el siguiente orden: análisis del problema, diseño del algoritmo para solucionarlo, codificación o implementación, compilación y ejecución, verificación, depuración, mantenimiento y documentación. Esta secuencia de pasos corresponde a los modelos *Software Development Life Cylce* o SDLC descritos en el sitio web Arkbauer (n.d.). Por otro lado, en el contexto de la investigación científica, Patrick J Mineault (2021) se refiere a un tipo de *programación estructurada* que describe y argumenta en su libro *The Good Research Code Handbook*. A partir de su propio caso, en el que el autor escribió código fuente durante el doctorado, propone que la programación debe enseñarse de manera estructurada. El autor cuenta que el proceso de escritura de código resultó complicado y frustrante, lo que le llevó a estudiar posteriormente programación de manera estructurada y bajo la óptica de las ciencias de la computación con lo que incluso logró colocarse en la industria de desarrollo de software.

La propuesta de Mineault de realizar un manual para programar de manera estructurada contrasta explícitamente con la aproximación exploratoria. Mineault señala que la enseñanza de programación enfatiza a veces la exploración, señalando los errores como oportunidad para aprender y alentando a que la mente corra libre. Esta teoría, continúa, sugiere que aprendes mucho sobre programación solo por el hecho de programar cada día. El autor no está de acuerdo

con lo anterior pues subraya que la exploración desestructurada es una forma muy ineficiente de aprender habilidades complejas como la programación. Esto difiere con la forma de programar en el live coding estudiada por Di Próspero y descrita por Knotts y Armitage quienes analizan y se pronuncian por dinámicas de prueba y error. Continuamente vamos a ver una tensión entre estas aproximaciones: una que apuesta por introducir a las personas al ámbito de la programación desde la exploración, la prueba y el error y otra que aboga por hacerlo de manera estructurada y desglosada.

Después de introducir ambas formas de programar, es posible decir que la programación prescriptiva estructura los procesos y tareas en objetos computacionales que pueden ser utilizados en la programación exploratoria del live coding. Esto implica una serie de diferencias, pero también similitudes y relaciones entre ambas formas. De esta manera, el código fuente, en el proceso de la programación prescriptiva, es encapsulado en un objeto computacional, programa o software que se integra a los recursos disponibles para hacer live coding. A partir de este acople parto para intentar ahondar en ambas perspectivas de programación, una exploratoria, que es a la que me he dedicado por varios años en el contexto de la creación musical con lenguajes de programación, y una prescriptiva que he intentado de diferentes maneras ante las sensaciones de desconcierto y frustración descritas por Mineault (2021). Lo anterior se explica como una resistencia del arte computacional enfocado en los procesos respecto al arte cuyo objetivo es llegar a un resultado terminado.

En el caso contrario, los modelos de programación de desarrollo de software y la programación estructurada no funcionarían si tratamos de aplicarlos en el momento de programar durante una presentación de live coding. Esto se puede observar en la práctica artística de Hiroki Matsui[17], quien programa de forma exploratoria en los diversos videos publicados en su canal de YouTube en los que no deja tiempo para pensar en procedimientos estructurados y jerárquicos de un modelo SDLC. Quizás, lo que resuelve en ese momento es el problema de la improvisación musical e implementa, o codifica, sus soluciones in-corporadas por la práctica de programación exploratoria continua que realiza.

17 Improvisación de Hiroki Matsui donde se puede observar una aproximación exploratoria a la programación de música: https://youtu.be/aw3UypG4TUc.

Con base en lo anterior, el artista que programa e investiga se enfrenta a una percepción externa de su quehacer que valida con sospechas su trabajo artístico, tecnológico y académico. En el ámbito artístico es visto como hacedor de herramientas e instrumentos, mientras que en el ámbito de desarrollo de software es considerado como programador carente de estructura y de procedimientos de desarrollo. Asimismo, surge la sospecha y resistencia en círculos académicos musicales hacia quien programa. En este ámbito se piensa que el artista que programa podría no estar habilitado como músico, compositor, instrumentista o investigador. Por otro lado, quienes incluyen la programación en sus procesos de creación artística, musical e investigativa piensan que la programación creativa es programación por derecho propio.

En este sentido, la reflexión que hice junto a Aarón Escobar en el capítulo *Agencialidad del código y del algoritmo: trazos sobre el ciclo práctica artística, desarrollo tecnológico e investigación* (Escobar y Villaseñor en García y Silva Treviño, 2022) da cuenta del papel de artista-programador-investigador con el que ambos lidiamos durante nuestras investigaciones. Aquí exponemos problemas del aprendizaje de máquinas y el live coding, desde una perspectiva artística, que nos orillan a desglosar nuestras figuras, marcos de referencia y prácticas artísticas. Lo anterior creo que sucede, en resonancia con la introducción de este capítulo, cuando abordamos temas de áreas especializadas. Con ello quiero reforzar la idea del párrafo anterior desde la que una actividad de investigación artística en tecnología musical enfrenta la dificultad de indagar un problema desde varios campos de conocimiento.

Cabe mencionar que la programación exploratoria y la programación prescriptiva muchas veces comparten los mismos lenguajes de programación, por lo que a primera vista se podría pensar que se trata de la misma forma de programar. La distinción entre ambas formas apunta a la metáfora de la barrera que hay entre estos dos conocimientos. Dicha barrera separa a quien realiza la práctica de live coding con un lenguaje de programación y a quien programa para desarrollar de dicho lenguaje. Ambos casos programan aunque con intenciones diferentes: uno se enfoca en la creación artística vinculada al live coding y el otro en el desarrollo de un objeto computacional para realizar dicha práctica. En las siguientes tablas coloqué algunas semejanzas y deferencias que, desde mi perspectiva, tienen estas dos aproximaciones.

| *Programación exploratoria* | *Programación prescriptiva* |
|---|---|
| Se realiza con un lenguaje de programación | Se realiza con un lenguaje de programación |
| Código fuente | Código fuente |
| Algoritmos | Algoritmos |

Tabla 1.1. Semejanzas entre la programación exploratoria y la prescriptiva.

| *Programación exploratoria* | *Programación prescriptiva* |
|---|---|
| explorar | estructurar |
| fallo y error | probar el código |
| modelos personales | modelos de desarrollo tipo SDLC |
| proceso | resultado |
| crear música | resolver problema |
| práctica artística | desarrollo tecnológico |

*Tabla 1.2. Diferencias entre la programación exploratoria y la prescriptiva.*

La relación que existe entre estas formas de programar es que los procesos artísticos de la programación exploratoria operan sobre los objetos resultantes de la programación prescriptiva, se informan entre sí. Esta relación posibilita que desarrolladores y artistas cambien su papel: los primeros también pueden crear música y los segundos también pueden desarrollar software.

*Apertura*

Una condición y postura personal para llevar a cabo el presente trabajo es la apertura del conocimiento tecnológico, científico y cultural, en el sentido de la apertura del código fuente de los programas propuesta por *la filosofía del software libre* y el *movimiento open source*, el acceso abierto a la ciencia u *open access* (Araiza Díaz en Azor et al., 2016; Babini. y Rovelli, 2020) y la cultura libre que permite compartir el trabajo artístico con diferentes licencias autorales como las *creative commons* (cc)[18]. De manera específica esto se ve reflejado en el caso de SonoTexto, el cual fue desarrollado con el lenguaje de programación SuperCollider desde donde establezco una

18 https://creativecommons.org/.

afiliación y posición por lo abierto. Esto se manifiesta en el uso de repositorios que ponen el código fuente de ese proyecto a disposición del público, en el uso de licencias libres y en la interacción con las comunidades de usuarios en foros y reuniones en línea.

Esta postura, y posibilidad, está permeada por el pensamiento que se desprende de las libertades postuladas por el software libre (GNU, 2021), la compartición de conocimientos computacionales de los *hackers* y el acceso al código fuente almacenado en repositorios como GitHub y GitLab bajo la lógica del código abierto. Esta investigación partió de la invitación y promesa de lo abierto del software libre, la ética hacker (Himanen, 2001) y el open source, lo que tomé para transitar de la práctica artística del live coding relacionada con la programación exploratoria a su desarrollo tecnológico relacionado con la programación prescriptiva.

Respecto a lo anterior, el software libre y el movimiento *open source*, aunque consideran al software un bien común, lo hacen de manera radicalmente diferente (Berry, 2004). El primero se enfoca en el derecho y libertad de utilizar el software con un carácter social que empodera al individuo, mientras que el segundo enfatiza la libertad de acceso al código en un contexto de mercado que potencia la eficiencia de desarrollo del software. Ambos términos pugnan por la apertura del código fuente aunque con objetivos diferentes que mantienen un antagonismo entre sus principales promotores y asociaciones como el caso de la *Free Software Foundation* (FSF) y la *Open Source Initiative* (OSI).

Por un lado, el software libre, según el sitio web de la FSF, promueve desde hace 35 años la libertad de quienes utilizan una computadora. Esto significa que usuarios y usuarias “tienen la libertad de correr, editar, contribuir y compartir el software” (Free Software Foundation, 2022). El sitio subraya que el software libre es una cuestión de libertad y no de precio. El discurso del sitio señala que el software libre se trata sobre tener el control de nuestras tecnologías pues considera a la computadora un bien comunal y no uno que pertenece a gobiernos y empresas que restringen y monitorean al individuo. Por su parte, el movimiento *open source* se desprende del desacuerdo entre miembros de la comunidad de software libre por lo que sus discursos tienen similitudes. El término *open source*, según el sitio web de la asociación *Open Source Initiative* (OSI), es acuñado en Palo Alto California en 1998 como una estrategia de lanzamiento del código fuente del navegador Netscape. Este suceso detona el inicio de lo que la OSI llama una

“oportunidad para educar y abogar por la superioridad de un proceso de desarrollo abierto” (Open Source Initiative, 2018).

David Berry, en su artículo *The contestation of code* (2004) hace una comparativa de los discursos de ambas corrientes. El autor plantea que las palabras claves que enfatiza el software libre en su discurso son el código, la libertad, los derechos, el poder, el progreso y la posición del sujeto. Por otro lado, el término *open source* denota una perspectiva de eficiencia técnica con un enfoque en el código, la propiedad, el mercado, la libertad y el individuo que lo vinculan a la apertura del código fuente desde una perspectiva neoliberal. Más allá de estas diferencias, el acceso al código fuente del software como bien común congenia con la postura de gestionar el acceso al conocimiento desde sus repositorios. Tal postura, desde las humanidades digitales y la ciencia abierta (Babini y Rovelli, 2020), considera a los repositorios de código fuente abierto un bien público y común que puede entrar a la categoría de los *datos abiertos de investigación*.

Por otro lado, Pekka Himanen estudió la esencia hacker. En su libro *The hacker ethic and the spirit of information age* (2001) recoge experiencias de personas que desarrollaron distintas tecnologías computacionales quienes coinciden, y abonan, que ser hacker es ver la programación como una actividad interesante y entretenida que plantea problemas. El autor encuentra que los hackers se aproximan a los problemas desde una actitud curiosa, entusiasta, juguetona y exploratoria. La mayoría se divierte con la resolución de pequeños retos de programación que posteriormente se convierten en desarrollos tecnológicos más elaborados como el caso del sistema operativo Linux o la encriptación tecnológica. Al igual que Franklin (1990) y Di Próspero (2015), Himanen ve en ello una visión del software y la tecnología como artesanías y la actitud manifiesta por un deseo de explorar. El autor compara este sentir con la actitud de la academia y otras actividades como el arte, la artesanía o la ingeniería, y sus casos reafirman que la actitud hacker no es exclusiva de la programación, sino que también está presente en actividades que parten de los mismos principios.

Para Hernando Barragán (2007) lo abierto implica revelar el código detrás de la obra digital ante la curiosidad de aquello que está detrás de su producción. Por su parte, Liliana Quintero (2013) plantea que el arte digital está vinculado a la tecnociencia, y debido a esto responde más a una cuestión económica que a una ética, lo que le impide activarse políticamente

en la esfera pública. Menciona que la tecnología está velada, lo que hace de la programación una actividad especializada, y esto hace al artista digital volverse experto, con lo que quizá la autora se refiere al desarrollo tecnológico de la obra digital. Quintero ve peligroso mantener oculta la tecnología y propone la filosofía hacker para desenmarañar lo técnico del arte y de esta manera mostrar qué hay detrás de la interfaz. Para ello, ve en el código una forma de mostrar lo oculto, con lo que aboga por una ética que permita visibilizar lo técnico y enfrentar el mundo de manera transparente contrario a hacerlo bajo el control del conocimiento de expertos.

Respecto a lo anterior, el proceso que implica involucrarse con la tecnología informática lo describe Irene Soria (2021) como un *devenir hacker*. La investigadora y educadora indaga las historias de mujeres que trabajan con tecnología informática en una capa en la que escriben código para resolver problemas, enseñar y administrar sistemas entre varias tareas. Soria diseña una gráfica con varias capas donde el punto más bajo está ocupado por la categoría “Mujeres Hacker: mujeres que desarrollan y hacen código” (p. 73). Bajo una metodología que incluye su propio devenir hacker, Soria nos relata cómo las hackers que ha entrevistado, así como ella misma, se han involucrado con la computación y la programación en una ruta en la que van conociendo “el interior de la caja cerrada” (p. 61). Dicho proceso comienza con adquirir ciertos saberes y experiencia para llegar en algún momento al nivel de la tecnología que considera más bajo. Este proceso, señala la autora, es posible por la apertura del código y el uso de software libre.

Conforme a lo anterior, la apertura presenta la oportunidad para acceder y compartir recursos de investigación, tecnológicos y artísticos. Es por ello que en este trabajo me posiciono a favor del código fuente, el conocimiento y la cultura declaradas como abiertas. Sin embargo, ante la bastedad del tema me enfoco mayormente en la apertura del código fuente del software libre. De esta manera, veo en la noción de apertura la posibilidad de vincular las dos formas de programación propuestas, así como los conocimientos, generalmente técnicos, que se encuentran en repositorios de código, manuales de los programas, libros de programación, tutoriales en línea, blogs, foros de usuarios y en el intercambio entre pares.

Con lo anterior no quisiera colocar el tema de la apertura desvinculado de un circuito más amplio que involucra la producción musical que apuesta por el uso de herramientas libres y

abiertas y por la distribución de sus resultados de la misma manera. Al respecto, Jorge David García (2021) propone lo anterior como un circuito de producción musical alternativo que incluye las dimensiones funcional, económica, estética, pedagógica y ética. Lo anterior, según García, basado en la noción de música libre de Ram Samudrala.

Bajo este esquema, la propuesta del presente trabajo se inserta en un panorama más amplio que el de la creación musical vinculada al desarrollo de software y la investigación. Con ello quiero señalar que este tipo de proyectos no ocurre de manera aislada, sino en circuitos en los que la creación musical y el desarrollo tecnológico se insertan junto a la producción, distribución y colaboración.

## 1.4 Giro y transposición

Dos conceptos que sirven para observar la relación entre práctica artística y desarrollo tecnológico en el live coding son *transposición* (Braidotti en Schwab, 2018) y *giro computacional* (Berry, 2011; Berry y Dieter, 2015). El primero se refiere a la movilidad entre conceptos de diferentes disciplinas y el segundo al cambio de perspectiva en la investigación académica de campos relacionados a las artes y las humanidades con el uso de tecnologías computacionales. Ambos conceptos ayudan a abordar la investigación del live coding desde una perspectiva de *investigación artística en tecnología musical*.

El libro *Transpositions: Aesthetico-Epistemic Operators in Artistic Research* (Schwab, 2018) retoma la noción de transposición desarrollada por la filósofa Rosi Braidotti quien usa el término en el contexto de la sociedad posindustrial. Braidotti describe tal noción de sociedad como fragmentada y dispersa que celebra la novedad y la globalización en la que prevalece la dicotomía de lo hipermoderno y lo arcaico. En ese contexto, la autora menciona que hay un avance del capitalismo que se aprovecha del desarrollo tecnológico para lucrar de manera injusta ante la apatía provocada por el neoconservadurismo político y liberal. Sin embargo, dice que aún persiste una resistencia ante semejante panorama ejercida desde la acción política colectiva.

Según Berry y Dieter (2015), tal resistencia afronta a la capacidad que tiene el capitalismo para desactivar el conflicto social en las sociedades posindustrial, tecnotrónica y del conocimiento. Este panorama lo confirma Grant H. Kester (2011), quien dice que el neoliberalismo económico en ascenso se ha dedicado a eliminar formas de resistencia pública y colectiva en favor del capital.

Para Braidotti, esas circunstancias exigen la necesidad de resistir, lo que se observa en una búsqueda de alternativas ante lo *posthumano*. Para ello, dice, se requiere un movimiento que apele por una subjetividad nomádica. Este comportamiento se da por la *transposición*, un concepto que la autora define como intertextual y situado en un cruce de fronteras o salto de código, campo o eje hacia otro de manera cualitativa y compleja. Braidotti recurre a una metáfora musical en la que la transposición corresponde a una variación armónica causada por cambios de tonalidad. En este proceso dice que se genera "un espacio en medio de zigzagueo y cruce: no lineal pero tampoco caótico; nomádico, pero responsable y comprometido; creativo pero también cognitivamente válido; discursivo y también materialmente embebido –es coherente sin caer en la racionalidad instrumental" (p. 26). Asimismo, la autora se basa en la teoría científica de la transposición de Barbara McClintock que consiste en un proceso de mutación genética no lineal. A partir de ello menciona que las transposiciones son múltiples y complejas y que ocurren en varios niveles a la vez. Menciona que se refiere a la movilidad y referencia cruzada entre disciplinas y niveles discursivos, como nociones que se transponen entre textos para generar su propio efecto. La transposición, dice, conecta comunidades separadas en su desplazamiento de zigzag de lo que está en tránsito.

En el contexto de la sociedad posindustrial descrito por Braidotti ocurre la idea de estar viviendo lo que Berry y Dieter (2015) llaman *giro computacional*. Los autores dicen que ese giro se manifiesta en un cambio de pensamiento que ocurre en diversos campos críticos, los cuales adquieren el compromiso de reflexionar sobre lo computacional como es el caso de las humanidades digitales, los estudios del software y los medios computacionales. Esto ocurre, continúan, con la introducción de sistemas de software en varios aspectos de nuestras vidas, los cuales producen cambios en nuestra forma de interactuar socialmente. Bajo este contexto, los procesos de investigación también sufren un giro que afecta sus prácticas. El propio Berry (2011)

señala que “la importancia de entender aproximaciones computacionales se refleja cada vez más a través de un número de disciplinas, incluidas las artes, las humanidades y las ciencias sociales, que usan tecnologías para cambiar la base crítica de sus conceptos y teorías –esencialmente un giro computacional” (2011 , p. 25).

Para Yuk Hui (2010), un giro suele atribuirse, desde las artes y las ciencias, a un cambio de época o forma radical de percibir el mundo. Dice que el giro computacional iniciado a finales del siglo XX parece hoy una obviedad y cuestiona si esto no querría decir simplemente que casi todo el mundo en occidente tiene una computadora o que muchas actividades cotidianas dependen de cómputo. A partir de ello argumenta que actualmente la forma de observar las dinámicas culturales corresponde a la lógica computacional, que no es más una herramienta sino una forma de ver la vida.

La transposición y el giro entre disciplinas, en el contexto de una investigación, requieren trasladar términos de un campo disciplinar a otro para detonar nuevos significados, incorporar sus formas de trabajo o fomentar cambios de paradigma. Esto se observa en la incursión que hacen las humanidades en las ciencias de la computación, donde hay un giro hacia tecnologías, metodologías y objetos de estudio computacionales que amplían las formas de producir conocimiento.

Para llegar a ejemplificar el concepto de transposición en el live coding primero partiré del problema de la dualidad que hay en la definición del *objeto sonoro* en la música concreta. Para ello me remito al libro *Tratado de los objetos musicales* de Pierre Schaeffer (1966/2008). En ese libro, el autor discute su investigación sobre la música concreta iniciada a mediados de los años 1940 en Francia. Una música creada con aparatos y técnicas de grabación y reproducción de sonido enfocada en la escucha de sonidos que han sido desplazados de su fuente original. Al inicio del libro Schaeffer menciona que la música concreta definía sus parámetros con términos de la acústica, por un lado, y de la música, por otro, en una especie de doble definición. A partir de ello dice que la música concreta se encuentra en el paso entre la ciencia y la música. Esto genera un problema en la definición de los objetos de pensamiento que lleva a un dualismo entre el objeto sonoro y las estructuras musicales detectado por Schaeffer. Para el autor, hay una ruptura entre formas de conocimiento de la ciencia y la música, por lo que su trabajo

busca conciliar con una relación transversal de las partes. Si bien Schaeffer no habla en términos de transposición, en su libro propone un recorrido temático en forma de zigzag que enlaza las cinco partes de su tratado: el hacer y escuchar, la física, la filosofía, el solfeo y la música. Esto es similar a lo que en términos de Braidotti sería, justamente, una transposición.

En el caso del live coding sucede algo parecido a lo que observó Schaeffer durante su práctica e investigación. En este caso, la dualidad del live coding ocurre entre las ciencias de la computación y las artes, cuyos campos tratan de definir una práctica artística computacional. Por ejemplo, términos de las ciencias de la computación como *tiempo real*, *patrón* o *iteración* hacen referencia a la *improvisación,* la estructura de un compás musical o a la repetición. A su vez, términos de la música electrónica como *sintetizador* son utilizados en la programación sonora para nombrar las unidades computacionales que generan sonido. Por otro lado, la creación musical, con o sin computadoras, es influida por procesos matemáticos que involucran el uso de algoritmos (Collins en Dean y McLean, 2018). Esto nos lleva a pensar en transposiciones entre el pensamiento artístico y el pensamiento computacional de procesos creativos atravesados por metodologías de las matemáticas aplicadas al campo de las ciencias de la computación.

La dualidad antes mencionada, poco obvia en el live coding, se encuentra implícita en la propia actividad de programación. La unificación de esta tarea en la escritura de código fuente, para que la computadora realice acciones, pierde los matices de las intenciones creativas y resolutivas. La formalización de la programación obstruye la posibilidad de brincar entre formas de conocimiento para expresar ideas artísticas y resolver problemas computacionales. Al señalar esta dualidad es posible buscar la relación transversal de las partes en la investigación de la práctica musical del live coding y apuntar a responder la pregunta sobre lo que se transpone entre la programación exploratoria y la programación prescriptiva. Se abre una posibilidad de conectar campos que están presentes pero que viajan de manera paralela y que, al torcer su trayectoria en la investigación, pueden converger. En este tipo de relaciones, el código fuente abierto es lo que Braidotti denomina un “concepto transportable”: un nodo que permite realizar transposiciones entre la música, las artes, las ciencias de la computación y la investigación en humanidades.

Pero, ¿es posible pensar un giro computacional dentro del campo del arte y la música computacionales? ¿es posible pensarlo dentro de la misma programación? Lo es en el sentido

que distintas investigaciones se interesan en la computación a un nivel más allá de lo que proponen las tecnologías. En este sentido, el giro computacional puede ocurrir entre modos de programación al pasar de un pensamiento de programación exploratoria del live coding a uno de programación formalizada y prescriptiva del desarrollo tecnológico. Esta propuesta puede parecer forzada pues el giro ocurre entre formas de programar y no entre metodologías de las humanidades y las ciencias de la computación. Pero, si pensamos al live coding como una actividad musical a partir de la que es posible generar conocimiento desde la investigación artística y a la programación estructurada como parte de una metodología de investigación entonces creo que es posible pensar en este giro.

Un ejemplo de transposición y giro computacional en una pieza de live coding se observa en el trabajo de Kate Sicchio (2019), quien programa sus coreografías con código. Sicchio propone que los paradigmas de programación imperativo y declarativo pueden aplicarse a patrones de movimiento en coreografías basadas en reglas. A partir de esto, la autora transpone conceptos entre las ciencias de la computación y la danza, los cuales aplica en su pieza *Moving Patterns* (2018)[19]. La propuesta de esta coreógrafa y live coder parte de la similitud que observa entre los paradigmas de los lenguajes de programación mencionados y la información que le transmite una coreógrafa a una bailarina para interpretar una coreografía, ya sea con instrucciones precisas o con un boceto de texto o imagen a partir del que improvisará. Aquí la noción de paradigma de programación cobra otro significado en términos de la coreografía de danza. Esta forma de pensamiento expresa lo que menciona Hui (2010) sobre la forma de ver la cultura a través de la lógica de la computadora que nos indica la presencia del giro computacional.

A simple vista esto crea una relación entre dos perspectivas, sin embargo, en el ámbito del live coding es señalada una “barrera” en continua tensión por derribarse que mantiene al usuario de un lado y al programador del otro. Para lidiar con ello, la figura del *artista-programador* ha sido introducida por diferentes autores (Di Próspero, 2017; McLean, 2011; Quintero, 2013; Rutz (en Schwab, 2018); Young, 2015) para referir un tipo de usuario que programa con un lenguaje de programación de manera creativa, quien se sitúa en el límite de esta

19 Extracto de la pieza Moving Patterns https://vimeo.com/278203530.

barrera. O, por el contrario, un programador-desarrollador que ha diseñado un software para expresarse. En ambos casos resulta una figura ambigua que cruza entre modos de programar, solo que en el caso del artista que usa un software este paso es borroso o no se logra, es decir, realiza una actividad de programación creativa pero no necesariamente una actividad de desarrollo tecnológico. Es aquí donde se realiza un giro que coloca a estas figuras del otro lado de la barrera. Pasar la barrera en un solo sentido es un primer paso, pero lograr un ir y venir nos coloca frente a una transposición computacional. El punto de cruce para lograr el movimiento de transposición se encuentra en la apertura del código fuente y del conocimiento. En este punto podemos oscilar entre ambos campos para generar conocimiento artístico y tecnológico. A su vez, mantener un movimiento de transposición permite cuestionar constantemente tal relación y no simplemente pasar de la forma exploratoria a la forma estructurada.

En este sentido, Carolina Di Próspero (2017) expone la subjetividad del live coder en cuanto artista programador. Menciona que el live coder dialoga con el lenguaje de programación y esto genera un híbrido entre lenguaje y artista en el momento de una presentación. Habla de los diferentes papeles que asume el live coder cuando busca quitar la barrera entre quien crea lenguajes de programación y quien crea música. Por otro lado, Liliana Quintero (2013) habla del grado de especialización que alcanza el artista cuando se enseña a programar y cómo esto le confiere conocimiento y responsabilidad que le permiten develar la tecnología detrás de sus piezas siempre y cuando esté dispuesto a deshacer las cajas negras alrededor. Por su parte, John Young (2015) cataloga a los compositores de música electroacústica en usuarios y desarrolladores, menciona que hay quien prefiere usar lo que ya está hecho mientras que otros se interesan por desarrollar sus propias herramientas. Lo anterior resuena con lo que refiere Marije Baalman (2015), quien habla del paso entre uso y desarrollo en el campo del live coding. La autora dice que dicho tránsito puede ocurrir cuando el practicante encuentra el límite de su expresión con un programa determinado y decide comenzar a desarrollar sus propias herramientas.

En el contexto de la investigación artística en tecnología musical lo anterior supone un doble giro. Uno de carácter computacional y otro de carácter epistémico en el pensamiento del artista que programa. El giro computacional lo transporta desde el uso de los lenguajes de

programación para hacer arte a su uso para hacer software, mientras que el giro epistémico lo lleva de los procesos tecnológicos y artísticos para crear arte digital a las metodologías de la investigación académica para producir conocimiento. Problematizar estos giros desde una perspectiva de investigación artística puede ayudar a clarificar cómo sucede, si no de manera general, sí en un estudio de caso que relaciona a la práctica artística con el desarrollo tecnológico.

Sin embargo, un giro hace un traslado de un punto A hacia un punto B que nos coloca en una posición diferente a la de inicio. En este caso se requiere un movimiento de ida y vuelta entre conocimientos y modos de hacer como el que plantea la transposición. Tal movimiento nos colocaría en medio de la figura que crea, programa e investiga. No obstante, el camino de la transposición encuentra una barrera fundada en el conocimiento tecnológico que impide el movimiento de ida y vuelta. Si la barrera cae es posible la transposición entre modos de programar y entre formas de producir conocimiento artístico, tecnológico y académico. Aquí, la apertura interviene y logra fisurar la barrera facilitando así el acceso a la creación artística digital, aunque esto no ocurre de igual forma para el acceso al desarrollo tecnológico pues demanda mayor conocimiento técnico.

# 2 Live coding: arte y música computacionales

La creación artística con lenguajes de programación es una forma de usar la computadora como medio de expresión. Esto se observa desde los primeros intentos por programar música a finales de los años 1950 y desde los primeros trabajos de arte computacional en los años 1960, cuando los algoritmos daban forma a la expresión gráfica. Varias décadas después, el uso de lenguajes de programación en la música y el arte se ve reflejado en nuevas prácticas computacionales entre las que se encuentra el live coding. A diferencia del primer arte computacional, el live coding se distingue por la escritura y ejecución de código fuente en el momento de una presentación. Esto es posible debido a factores tecnológicos y culturales como el desarrollo de procesadores más rápidos, lenguajes de programación que traducen al vuelo el código fuente en código máquina, el cambio de perspectiva en el diseño de software musical, el acceso a una gran cantidad de conocimiento y el intercambio de información entre individuos y comunidades de práctica facilitados por el internet.

El live coding, como práctica artística, se conforma a principios de los años dos mil, inicialmente con una orientación musical y, posteriormente, visual, literaria y coreográfica. En el caso musical, la línea histórica que sigue está poco ligada a la música electroacústica de posguerra, a diferencia de otras prácticas musicales y sonoras que trabajan con tecnologías de grabación y con tecnologías digitales. Esta práctica más bien proviene de un ámbito del arte computacional ligado a las matemáticas, la ingeniería y las ciencias de la computación. Asimismo, proviene de la inquietud que se da en este último círculo por compartir conocimientos informáticos bajo la llamada *ética hacker* (Himanen, 2001; Levy, 2010).

Parte de los procesos creativos del live coding, en contraste con la música electroacústica, tienen que ver con la naturaleza escrita de las prácticas artísticas basadas en programación textual. Tales prácticas relacionan al practicante de live coding con sus materiales musicales y sonoros desde la notación simbólica del código, mientras que la música electroacústica lo hace principalmente desde la señal de audio (Larrieu, 2019; Magnusson, 2019; Keislar, 2012).

Lo anterior es un panorama general de lo que será desarrollado en este capítulo, mismo que introduce la relación entre la práctica artística y el desarrollo tecnológico en el live coding desde una línea histórica trazada en sus tres primeras secciones. Tal recorrido inicia con el arte computacional a mediados de los años sesenta, luego revisa algunos antecedentes de la música por computadora para llegar hasta el año 2004, cuando el colectivo Toplap escribe el manifiesto sobre el live coding. Luego, la cuarta sección describe algunos desarrollos actuales de software dedicados al live coding, concretamente la plataforma Sema, el secuenciador CaosBox, la librería INSTRUMENT, la clase MIRLC y la plataforma para live coding visual Hydra. Para concluir, la quinta y última sección señala el interés por el estudio del live coding desde diversas disciplinas como las ciencias de la computación, las humanidades y la ciencias sociales para plantear problemáticas relacionadas con la práctica artística del live coding.

## 2.1 Arte y tecnología en el arte computacional

Hacer arte con computadoras es una práctica relativamente nueva. Algunos autores (Candy et al., 2018; Reichardt, 1969) señalan la mitad de los años 1960 como el inicio de la exposición de los primeros trabajos de arte computacional. En ese momento, la computadora, en su forma de unidad central o minicomputadora (Higgins y Khan, 2012)[20], comenzaba a ser utilizada por científicos y artistas para la creación artística, lo que la convirtió en un nodo entre diversas prácticas y disciplinas como la música, la poesía, la escritura y las artes visuales. Desde el inicio, las prácticas de arte computacional han requerido la programación y, aunque la mayoría de las actividades realizadas hoy en día con una computadora se llevan a cabo con una interfaz gráfica, hay un regreso al trabajo artístico con lenguajes de programación. Lo anterior se puede ver en casos como el uso del microcontrolador Arduino, el software Processing o los diferentes lenguajes de programación dedicados al live coding, así como en las tecnologías de hardware utilizadas ampliamente en el arte digital entre las que se encuentran las computadoras de placa Raspberry Pi y Bela. Esto es algo que se observa en términos como el de *software art* –creado

20 Higgins y Kahn, (2012) se refieren a las unidades centrales o *mainframes* que en los años 1960 y 1970 tenían la dimensión de un cuarto y a las minicomputadoras operadas con transistores.

como categoría por los festivales Transmediale y Ars Electronica (Barragán, 2012)– que hace alusión a que el código computacional es considerado el material artístico.

Paul Crowther (2019) ubica el surgimiento del arte realizado con computadoras bajo el término *arte digital* en la transición de los años 1950 a 1960 en la era posindustrial. Este autor argumenta que la computadora se encuentra en el centro del cambio económico de ese momento que determinó la forma de producción y distribución de los bienes. En ese entonces, dice, la cibernética y la teoría de la información surgen en el paso de la edad moderna a la posmoderna que trae consigo la formación de la sociedad de la información. En tal paso, continúa, las prácticas artísticas se vuelven plurales y el arte como concepto es validado. Asimismo, la computadora como tecnología se convierte en un medio artístico que va a demandar al artista nuevas habilidades informáticas. Por el contrario, Higgins y Kahn (2012), discuten la poca atención que ha recibido el primer arte digital, que ubican entre los años 1950 a 1970, por parte de la historia del arte y de los medios. Los autores argumentan que este desinterés, e incluso menosprecio, se debe a la falta de un aparato crítico para entender lo que se produjo en esa época. También creen esto se debe a que el primer arte creado con computadoras es considerado malo o inmaduro y es relacionado con una aspiración del ingeniero que quiere ser artista o viceversa. Ello, dicen, ha inhibido la aceptación de que la tecnología informática se haya fusionado con una práctica artística. Lo anterior, como veremos más adelante, se va continuar repitiendo en los años 2000 con prácticas artísticas digitales que entran a la escena del arte y la música como es el caso del live coding.

Según Candy, Edmons y Poltronieri (2018), la primera exposición de gráfica por computadora fue organizada por Max Bense en enero de 1965, en la galería *Studiegalerei* de la Universidad de Stuttgart. En esa exposición se presentó el trabajo de Georg Nees, el cual consistía en una serie de dibujos creados bajo las instrucciones de un programa computacional, desarrollado por el propio autor, que dirigía el brazo de un plóter para realizar trazos sobre papel.

La programación para controlar un plóter en la creación de gráfica y dibujo se hizo común en ese periodo como se observa en los trabajos de Vera Molnár y Roman Verostko. En el caso de Georg Nees, el artista no trazaba de manera directa en el papel, sino que ideaba el trabajo desde la programación del software o el diseño de algoritmos que instruían al plóter para dibujar.

Roman Verostko, pionero en el dibujo por computadora y miembro del grupo de los *algoristas* –artistas que trabajan con algoritmos para producir formas de arte– explica en el ensayo *Epigenetic Art Revisited: Software as Genotype* (2003) la metodología y la tecnología con la que realiza sus pinturas algorítmicas. Verostko se pregunta si un artista puede codificar arte y si es posible escribir instrucciones con tal fin, a lo que responde afirmativamente. Plantea que un dibujo realizado con procedimientos algorítmicos se incorpora a la sensibilidad de la mano bajo el concepto *mente-mano*: la presencia de la mano en la mente del *algorista* cuando realiza dibujos con procedimientos algorítmicos. Menciona que al escribir código se añaden cualidades estéticas y se establecen límites como el tamaño y material del papel o la receptividad de la tinta. Para trasladar el código creado por la mente-mano a un dibujo, el autor requiere operar una computadora que instruye a una máquina de dibujo *multipen plotter*, a la que añade diferentes lápices y brochas. Verostko menciona que el brazo del plóter no es un simulador del brazo del artista, sino que ejecuta la mente-mano de este, la cual excede el alcance de su mano física.

Tres años después de las primeras exposiciones de gráfica por computadora, en 1968, Jaisa Reichardt organizó en Londres la exposición *Cybernetic Serendipity*, que mostró diferentes piezas de arte computacional creadas por artistas, ingenieros y científicos. La temática de la exposición giró en torno a la cibernética, en la que se expusieron gráficas y animaciones generadas por computadora; música compuesta y tocada por computadoras; así como poemas y textos de computadora, aparatos cibernéticos, ambientes cibernéticos, robots a control remoto y máquinas de pintura. La exposición, cuenta Reichardt (1969), muestra relaciones entre tecnología y creatividad, unas formas creativas engendradas por la tecnología, en la que participan artistas en la ciencia y científicos en el arte. Lo anterior da cuenta de una aproximación al arte computacional desde diferentes perfiles y disciplinas. Reichardt hace énfasis en la definición del artista computacional, quien no proviene necesariamente de una escuela de artes, sino que puede venir de las ciencias, la tecnología o cualquier otro campo.

Frieder Naker (en Candy et al., 2018) cuenta que la primera exposición de gráfica digital –aquella de Georg Nees– fue organizada por Max Bense, quien aquella ocasión acuñó el término *arte artificial* para distinguir entre el arte hecho por humanos y el producto de la computadora. En este sentido, Miguel Carvalhais y Pedro Cardoso (2018) mencionan que se han utilizado

diferentes términos para hacer referencia al arte hecho con computadoras, lo cual consideran innecesario, pues plantean que el término *computación* describe la prioridad de los procesos sobre los datos en el arte que se expresa con una computadora. Además mencionan que el arte computacional no necesariamente reside en una computadora digital o en un algoritmo sino en la "computación como un medio para describir estructura y proceso" (p. 221). De esta manera, definen al arte computacional como aquel basado en el procesador y hacen una distinción entre el uso de una computadora para guardar datos y el uso para procesarlos. El propio Carvalhais (2022) cuenta que algunos artistas que no tenían acceso a un computadora incluso realizaban los cálculos de un algoritmo a mano para trazar sus obras. Esto, propone, los convertía en computadoras humanas, como el caso de varias piezas realizadas por la artista pionera Vera Molnár antes mencionada.

Otra categoría que engloba al arte con computadoras es el término *arte de los (nuevos) medios*, categoría que impera en el discurso de los congresos International Symposium on Electronic Art (ISEA) y Media Art History (MAH). Los archivos de estos congresos revelan la aproximación a diferentes formas de arte que utilizan la computadora. Por su parte, Edward Shanken (2005), se refiere a la categoría *arte, ciencia y tecnología*, que engloba a la anterior. Este discurso de intersección entre diferentes campos se ve expresado en el subtítulo del Festival Ars Electrónica, organizado en Linz desde 1979, que apunta a la relación arte, tecnología y sociedad, con ello abarca un abanico de prácticas artísticas en las que el uso de la computadora tiene un papel preponderante. En este escenario, el término *arte computacional* no aparece mencionado tal cual, se diluye como una subcategoría implícita en las categorías de arte digital, arte electrónico y arte de los nuevos medios.

Quien ha trabajado este término es el diseñador, músico y académico Miguel Carvalhais quien en su libro *Art and Computation* (2022) define el arte computacional como aquel que utiliza procesos y algoritmos que pueden ser computados tanto por humanos como por máquinas. Para Carvalhais el arte computacional no es aquel que usa una computadora y sus procesos "como remediadores o simuladores de medios y herramientas" (p.16), sino aquel que establece una relación estrecha con la computación, la cual "es fundamental para su entendimiento, para la

construcción de significado y para el desarrollo de esa experiencia intersubjetiva a la que llamamos arte" (p. 16).

Por otro lado, en la revisión histórica sobre los archivos de los festivales y congresos mencionados, México no figura como país gestor de arte digital. Esto tiene que ver con lo que indican Reynaldo Thompson y Tirtha Mukhopadhyay (2021) acerca de la falta de catalogación y conservación rigurosa del arte digital en Latinoamérica[21]. Lo anterior coincide con el señalamiento de Monreal (2020) sobre la ausencia de sistematización de los archivos de los centros mexicanos dedicados a estas expresiones. Respecto a ello, Thompson y Mukhopadhyay (2021) señalan que el factor económico es determinante pues permite a museos y centros de países altamente industrializados acceder a grandes presupuestos y patrocinios para operar una serie de eventos y exposiciones con su respectiva catalogación. Prueba de lo anterior es el repositorio en línea *Other New Media Art Archives* de ISEA que pone a disposición diversos archivos de festivales, congresos, galerías, bienales, fundaciones, museos, universidades, centros e institutos en los que solo figura la bienal de São Paulo[22].

El arte computacional se ha definido por la introducción de la computadora y un pensamiento computacional a diferentes disciplinas artísticas. En un inicio fueron científicos quienes utilizaron la computadora para expresarse, lo que fue descalificado tanto desde el arte como desde la tecnociencia. Por un lado, la gráfica generada con algoritmos ejecutados en una computadora mostrada en las primeras exposiciones de arte computacional no calificaba como arte para los artistas asistentes y, por otro lado, para las ciencias de la computación el código que busca explorar de manera creativa, contrario a resolver un problema, no es tomado en cuenta en muchos discursos tecnológicos (Marino, 2020). Las primeras aproximaciones a la creación de obras de arte con una computadora han requerido entender los lenguajes de programación y los procesos de computación como medios de expresión artística. En el caso de la música por computadora, esta se ha desarrollado de manera paralela y con una menor proyección; no obstante, fue incluida como categoría desde la primera exposición de arte computacional

21 Un esfuerzo importante por resguardar y catalogar el arte de los nuevos medios en México es el Centro de documentación Príamo Lozada (CDPL) promovido en un inicio por Tania Aedo dentro del Laboratorio de Arte Alameda (LAA) en la Ciudad de México desde 2015. Sin embargo, al entrar al sitio web de LAA no aparece algún vínculo hacia ese archivo que nos proporcione información.

22 Véase *Other New Media Art Archives* de ISEA: https://www.isea-archives.org/other_new_media_art_archives/.

*Cybernetic Serendipity*. La siguiente sección vincula el inicio de la música por computadora con el live coding a través de una serie de desarrollos tecnológicos e históricos.

## 2.2 Música por computadora y su relación con el live coding

Programar música con un lenguaje de programación conlleva un acto creativo en un medio con características propias. En este sentido, entiendo la música por computadora desde las posibilidades de su medio digital y no como una traslación de un medio analógico a uno digital que lo sustituye, simula o representa.

El espectro de la música por computadora es amplio en cuanto a géneros musicales y tecnologías empleadas. En tal variedad de prácticas artísticas y desarrollos tecnológicos algunos autores han definido el *instrumento musical digital* compuesto por la computadora y el software. Esta sección del capítulo comienza con algunas definiciones de dicha categoría para, posteriormente, dar paso a una línea histórica sobre la música por computadora que conecta con la práctica del live coding.

Thor Magnusson, conocido live coder e investigador, expone en su libro *Sonic writing: Technologies of material, symbolic and signal inscriptions* (2019) la relación entre la música, los instrumentos musicales, sus tecnologías y su forma de notación a lo largo de la historia. El autor argumenta que desde el inicio de la música, los instrumentos musicales presentan la inscripción de su tecnología en sus cuerpos, lo que nos permite suponer cómo funcionan. El autor expone cómo ello ha incidido en la forma de pensamiento musical de cada época y cómo en la actualidad la aproximación a lo digital está transformando nuestra forma de pensar la música y sus prácticas. Según Magnusson, el instrumento musical digital ha pasado de la imitación de los instrumentos electrónicos y acústicos a una forma propia desde la implementación del audio en tiempo real. Este autor menciona que la computadora como instrumento musical digital se caracteriza por ser de una naturaleza algorítmica, tener una facilidad de automatización y mapeo y por su potencial de aprendizaje.

Por su parte, el profesor y tecnólogo musical Andrew Brown (2007) en el libro *Computers in music education: Amplifying musicality*, propone que la computadora puede verse como *herramienta*, *medio* e *instrumento* cuando es usada para la creación musical. El autor confiere cada perspectiva a una metáfora que ayuda a entender la computadora cuando es introducida a la práctica musical en un contexto educativo. Brown dice sobre esta tecnología que "cuando es vista como herramienta, la computadora es tomada como un dispositivo para ser controlado, cuando es entendida como medio esta se convierte en un vehículo para explorar posibilidades musicales y, cuando es abordada como instrumento, puede ser un conducto para la expresión musical" (2007, p. 6).

A su vez, Sergi Jordá (2005), reconocido desarrollador e investigador en el ámbito de la tecnología musical, emplea el término *nuevos instrumentos musicales digitales* para referirse a instrumentos basados en computadoras. El autor hace énfasis en la construcción propia de instrumentos para la improvisación musical, lo que llama *laudería digital*. Tales instrumentos digitales, propone el autor, involucran en su construcción factores tecnológicos y humanos como la electrónica, la programación, el procesamiento de señal, la psicología, la fisiología, las interfaces humano-computadora, el mapeo y la música. Jordá menciona que la laudería digital es un trabajo de construcción en continuo proceso que pone al laudero improvisador en un bucle que relaciona la imaginación, el diseño, la manufactura y la improvisación con la creación musical. El autor resalta que uno de los objetivos de crear nuevos instrumentos musicales digitales es suscitar músicas antes no escuchadas que se generan a partir de nuevas técnicas, sean estas composicionales o instrumentales.

La música realizada con una computadora ha sido catalogada de diferentes maneras. Una de estas categorías es la *música por computadora* o *computer music*. Según Douglas Keilsar (2017), el término es introducido en el siglo XX con un significado ambiguo y polifacético que a veces es intercambiado con el de música electrónica y música electroacústica. Keislar detecta dos acepciones para su estudio, uno confiere a la música por computadora la cualidad de género musical y otro la de disciplina técnica. El campo incluye diversas prácticas, algunas de ellas ligadas a la creación musical con lenguajes de programación, entre las que se encuentran la composición algorítmica (Collins, 2018), la música generativa (Nierhaus, 2009), la síntesis

digital de sonido, la recuperación de información digital (Xambó et al., 2019), la escucha y el aprendizaje de máquinas (Knotts y Paz, 2021) y el live coding.

La primera presentación musical hecha con una computadora, según Brown (2007), ocurrió durante el Congreso de Computación de Australia en agosto de 1951 cuando Geoff Hill y Trevor Pearcey programaron música en la computadora CSIRAC, la cual estaba diseñada para fines científicos. Pearcey (en Doornbusch, 2009) cuenta en una entrevista realizada en 1996 que esto fue posible porque la computadora contaba con una pequeña bocina que producía tonos en forma de pulsos para alertar al programador cuando un programa no corría bien, lo cual se notaba cuando el patrón rítmico que producía se alteraba. Pearcey y Hill aprovecharon este componente para producir los tonos de una escala diatónica en el rango de dos octavas y programarlos para recrear algunas melodías de canciones populares. Keislar (2012) agrega que este suceso ha pasado desapercibido para la comunidad de música por computadora pues en ese momento no había gente involucrada en la composición musical con esas técnicas.

Por su parte, Magnusson (2019), menciona dos acontecimientos que marcaron el inicio de la creación de música por computadora: la *Suite Illiac* y el lenguaje de programación MUSIC I. La composición de la *Suite Illiac* fue realizada en 1957 por Lejaren Hiller y Leonard Isaacson para un cuarteto de cuerdas con el uso de la computadora ILLIAC I, con la que generaron varias secuencias de notas musicales que posteriormente transcribieron a una partitura. Este caso resulta interesante pues aunque la partitura es resultado de un proceso computacional algorítmico la generación del sonido la producen los músicos de un cuarteto de cuerdas leyendo la partitura. Aquí se puede traer a colación el concepto *mente-mano* de Verostko, ya que esta suite fue conceptualizada de manera similar a sus dibujos algorítmicos (véase § 2.1). Es decir, la partitura es una extensión de la mente del compositor, quien escribe los algoritmos para generarla –de hecho Versotko también propone el concepto *mente-oído*, el cual ejemplifica con la notación musical de una partitura que funciona como el código o algoritmo que instruye a un intérprete para tocar su contenido. Por otro lado, la suite es un ejemplo de un proceso computacional que se genera, más no reside, en la computadora para su ejecución. Este tipo de obras computacionales es problematizado por Carvalhais y Cardoso (2018), quienes argumentan que estas piezas se

consideran en el ámbito de la creación computacional por su enfoque en el procesador de la computadora y no por los datos que residen en ella.

El trabajo de Hiller e Issacson irrumpió en una comunidad musical escéptica del uso de la computadora para la composición. La pregunta de la que parte Hiller (en Reichardt, 1968) es si una computadora puede usarse para componer una Sinfonía. La pregunta es formulada en términos culturales más que técnicos. Esta refleja una crítica al pensamiento de la época acerca de la composición que, según Hiller, seguía arraigada a la idea romántica de la inspiración. Hiller arremete con una respuesta pragmática que exalta las posibilidades de un análisis objetivo, cuantificable, racional y matemático de la música a través de la teoría de la información y de la investigación acústica de la música. El autor argumenta que el análisis musical en estos términos observa las relaciones del sonido, definido físicamente, y el tiempo que recurre a las matemáticas para observar el contenido y la combinación de símbolos y números.

El mismo año 1957, el ingeniero en electrónica Max Mathews desarrolló en la compañía de telefonía Bell Labs el primer software de síntesis sonora llamado MUSIC, a los que seguirían diversas versiones. Keislar (2012) atribuye a Mathews el descubrimiento de la síntesis digital de sonido, la cual define como “la construcción numérica de sonidos ‘desde cero’” (p. 19). Ese acontecimiento, según el autor, pone a la computadora como una fuente generadora de sonido a diferencia de un reproductor de sonido grabado digitalmente. Mathews encuentra por primera vez la forma de producir sonido desde el hardware de la computadora y controlarlo con software. Además, agrega el autor, MUSIC aporta la idea de *unidad generadora*, que consiste en pequeños módulos que se conectan entre sí para generar o procesar sonido. Algo que señala Keislar es que esta serie de lenguajes no estaban enfocados en la programación algorítmica en sí, lo que va a ocurrir con lenguajes de programación desarrollados después que heredaron el concepto de unidad generadora como Nyquist (1997), SuperCollider (2002) y Chuck (2003).

Los Laboratorios Bell jugaron un papel importante para la música por computadora, pues además de apoyar el trabajo de Mathews, acogieron compositores para trabajar en colaboración con sus ingenieros. Es el caso de James Tenney (en Reichardt, 1969) quien cuenta sobre su paso entre los años 1961 y 1964, en los que tuvo la oportunidad de trabajar en la composición de piezas de cinta con sonidos generados con computadora. El compositor, resalta la aceptación

paulatina de sonidos sintéticos en su trabajo al pasar por un proceso de aprendizaje para entender mejor la física del timbre en este tipo de sonidos. A diferencia de Hiller quien se enfoca en las estructuras, Tenney ahonda en un área igualmente física que se relaciona más con las posibilidades tímbricas del sonido.

La sección dedicada a las presentaciones históricas de live coding del sitio wiki de Toplap (s.f.), colectivo que difunde la práctica desde el año 2004, propone una línea temporal en tres partes que conecta al live coding con eventos relacionados a desarrollos informáticos. La conexión temporal está marcada por los antecedentes del live coding situados en los años 1960 antes de la tecnología de traducción implementada en los llamados *lenguajes interpretados*. Los ejemplos de esta etapa del live coding previa a los lenguajes interpretados, presentados por el colectivo, son de carácter ficticio e irónico. Con ello simplemente establecen una fecha histórica que va a hacer posible el live coding debido a un desarrollo tecnológico dedicado a la traducción de código entre humanos y máquinas.

La tecnología de los lenguajes interpretados, según Joyanes Aguilar (2019), permite convertir las instrucciones que escribimos en forma de código fuente a el lenguaje de máquina. Esto se lleva a cabo con el uso de un programa traductor llamado *intérprete* que se encarga de traducir una a una las instrucciones escritas. Como deja ver el sitio wiki de Toplap, los lenguajes de programación interpretados son un desarrollo tecnológico muy importante para el live coding, ya que este tipo de lenguajes permiten escribir y modificar el código al vuelo, línea por línea. Esto quiere decir que el código fuente que describe las acciones puede ser modificado, y ejecutado, sin la necesidad de parar el programa para compilar cada nuevo cambio. Lo que se consiguió con esto fue mantener el continuo de la generación de sonido en el contexto de la reescritura de código fuente que ocurre de manera discontinua.

La segunda parte de la línea histórica presentada por Toplap está situada en los años 1980. Este momento lo vinculan al lenguaje de programación FORTH, el cual para ese entonces ya tiene implementada la tecnología de traducción interpretada en su diseño. En ese momento se llevan a cabo algunas presentaciones mediante la modificación de código en vivo con dicho lenguaje de programación. Una de estas fue aquella realizada por Ron Kuivila en 1985 en el STEIM de Amsterdam, la que el colectivo considera que fue la primera presentación

documentada de live coding. Por su parte, Ward et al. (2004) ubican en el periodo de los años 1980 la práctica del ensamble de computadoras The Hub, quienes también utilizaban FORTH para crear música por computadora en red. Los autores comentan que este ensamble permitía a los asistentes a sus conciertos acercarse a ver cómo programaban, en uno de los primeros intentos por mostrar la actividad del artista-programador al público.

Por último, la tercera parte de la ruta histórica propuesta por Toplap, marcada como la era de la proyección del código, señala que las presentaciones de live coding de la banda de computadoras Slub y de Julian Rohrhuber en el año 2000 pueden considerarse las primeras que documentan la proyección del código para la audiencia. Esta sección termina en el año 2007 con la referencia de diferentes presentaciones y eventos de live coding de las que destaca la formación del colectivo en el año 2004.

A primera vista parece que el live coding no viene de la primera música por computadora comenzada con la serie de lenguajes de síntesis sonora MUSIC-N. Más bien aparece como una actividad musical dentro de círculos de programadores y artistas entusiastas de la tecnología informática que utilizan lenguajes de programación científicos como FORTH o de propósito general como Perl. Esta aparente discontinuidad quizá tiene que ver con la inclusión de la música por computadora como una categoría de la música electroacústica (Larrieu, 2019), campo que se enfoca en la creación musical desde los procesos y metodologías que usan la señal de audio (Young, 2015) y no desde la notación o el código computacional (Magnusson, 2019). Esto ha mantenido a la música por computadora como categoría de perfil bajo dentro de la música electroacústica. Por ejemplo, el proyecto *ElectroAcoustic Resource Site* solo menciona una entrada relacionada con el término y carece de bibliografía ligada al live coding[23]. No obstante, la música por computadora, fuera de ese contexto, tiene difusión como práctica autónoma lo que se observa con la publicación de la revista *Computer Music Journal*[24] editada trimestralmente desde 1977 por MIT Press, cuyo número de 2014 está dedicado a la práctica de live coding[25]. Más aún,

23 http://www.ears.dmu.ac.uk/.
24 https://direct.mit.edu/comj.
25 Es interesante observar la sección *Most Cited* del COMJ [recuperada de Wayback Machine con fecha del 31 de mayo de 2020]. En ella podemos ver el conteo, según la captura de esa fecha, de los artículos más citados en los que el tema live coding aparece solo una vez. [Información consultada el 18 de octubre de 2021]. https://web.archive.org/web/20200531192248/https://www.mitpressjournals.org/action/showMostCitedArticles?journalCode=comj.

el live coding encuentra mayor resonancia como categoría de la música algorítmica (Dean y McLean, 2018).

La conexión del live coding con la música por computadora puede trazarse desde una línea retrospectiva que conecta diferentes lenguajes de programación musical con el software SuperCollider. Para ello se puede partir de la idea de UGen o unidad generadora de sonido desarrollada por Mathews en 1960 en MUSIC-N, la que fue incorporada en el diseño de diversos lenguajes de programación musicales que le siguieron como es el caso de SuperCollider. Aunque originalmente este software, creado por James McCartney en 1996, no fue diseñado para el live coding, su cualidad tecnológica aporta desde el inicio extensiones desarrolladas para su realización. Por ejemplo, la librería JITLib o *just in time programming* diseñada para realizar live coding o, años después, la clase SuperDirt que ayuda a gestionar el sonido de TidalCycles, software escrito por Alex McLean y quizá el más utilizado en el live coding sonoro. El desarrollo de ambos objetos computacionales (JITLib y SuperDirt) fue realizado por Julian Rohrhuber, pionero del live coding y miembro fundador del colectivo Toplap y de la banda de laptops PowerBooks UnPlugged. La librería JITLib de SuperCollier, creada a inicios del año 2000, aprovecha la capacidad de los lenguajes interpretados para modificar un programa mientras está funcionando. Tal capacidad, como mencioné antes, transformó la creación de música con computadoras de lenguajes de programación como SuperCollider, ChucK o Perl al pasar de la escritura de código computacional en tiempo diferido, ligada a la composición, a la escritura en el momento que permite la improvisación con código.

Aunque en un inicio el desarrollo de SuperCollider fue con fines personales y poco después estuvo a la venta, desde el año 2002 este software fue registrado como software libre bajo una licencia GNU General Public License. A partir de entonces su código fuente ha estado abierto, no tiene costo y es mantenido por una comunidad de desarrolladores. Según el prefacio del *SuperCollider Book* (2011), escrito por el propio McCartney, la primera versión de SuperCollider (1996) tiene su origen en dos piezas de software que escribió para uso personal: el programa Synth-O-Matic y el objeto Pyrite escrito en MAX. La primer versión, cuenta el autor, tenía una sintaxis parecida a C y Scheme, la cual cambió para la versión 2 a una más parecida a SmallTalk que es la que continúa usando la parte de sclang hasta hoy. Mientras McCartney

trabajaba en la versión 3, dice que decidió hacerlo open source para que siguiera creciendo y todo mundo tuviera acceso a modificarlo. A partir de entonces el software ha generado comunidades de usuarios, un simposio anual (2006 - 2013), artículos académicos, manuales en diferentes idiomas, foros y encuentros locales.

Pero, ¿qué tanto comparte el live coding con el arte computacional y la música por computadora? Por un lado, algunas similitudes emparentan al live coding con el arte computacional como la entrada a los circuitos de arte desde las ciencias, el rechazo del sector artístico en su inicio, el discurso de la relación entre arte y tecnología y los perfiles heterogéneos de sus practicantes. Por otro lado, es posible considerar al live coding como una práctica de música por computadora, aunque esta definición restringe su carácter que abarca múltiples prácticas, pues se enfoca solo en lo sonoro. Esta pregunta también tiene respuesta en los circuitos en los que circula el live coding que, más que contener similitudes con el arte computacional y la música por computadora, se diferencian por los principios de apertura a los que se adhieren.

Además de la línea que conecta MUSIC-N con SuperCollider, este último provee a otros lenguajes dedicados al live coding su motor de audio o servidor como es el caso de Sonic Pi y FoxDot o sus recursos de gestión de sonido como el mencionado caso de TidalCycles. De manera paralela al concepto de live coding el desarrollo del programa ChucK (2003) introdujo el término programación *on-the-fly* para nombrar la programación de música en vivo. Los lenguajes mencionados han sido específicamente desarrollados para la creación de música, aunque también algunos lenguajes del ámbito científico como Forth y Perl han sido utilizados para la creación musical.

No está claro si la ruta histórica presentada hasta aquí es la que sigue el desarrollo del live coding, pues, no es sino hasta cincuenta años después de la aparición de dichos desarrollos tecnológicos que comienza a definirse. Lo que se puede rastrear es que esta práctica viene de círculos de programadores afines al software libre, al movimiento hacker y a las músicas experimental y electrónica bailable o *electronic dance music* (EDM). Y aunque aparentemente el arte computacional y la primera música por computadora no son percusores directos del live coding, este último se aprovecha de la línea de desarrollos tecnológicos que la computación aportó a la música por computadora. Por su parte, el live coding originó un cambio de paradigma

en la música computacional que fue expresado en el manifiesto escrito por el colectivo de artistas programadores Toplap en el año 2004, el cual define las características y posturas de la práctica de live coding como veremos en la siguiente sección.

## 2.3 La definición del live coding según el Manifiesto Toplap

El término live coding comienza a usarse en el año 2003 para referirse a una práctica de programación musical. Sin embargo, esta práctica había comenzado a configurarse tiempo atrás con la llegada de los *programas intérpretes* del código fuente que posibilitaban modificar un programa sus procesos para compilarlo. En el año 2004, Julian Rohrhuber y Renate Wieser organizaron el simposio Changing Grammars – A Live Audio Programming Symposium en la Escuela Superior de Artes Visuales de Hamburgo. Este simposio fue el primer encuentro académico que reunió a varios programadores y músicos que en ese momento trabajaban con la programación en vivo para generar sonido. La temática del simposio proponía “mudarse de la idea del programa como herramienta hacia la idea del código como conversación, un discurso científico y poético acerca del fenómeno acústico, su percepción y su descripción” (HFBK, 2004).

Durante este simposio se formó el colectivo Toplap, cuyos integrantes escribieron un manifiesto en el que postularon los puntos de la práctica de live coding y formalizaron el término en un contexto musical y sonoro. Dicho manifiesto, escrito de manera informal y con un tono irónico, es un punto de referencia para entender las prácticas y las líneas de investigación que se han conformado desde su publicación. El manifiesto, aunque es un referente dentro de la comunidad de live coding, fue publicado como borrador o *ManifestoDraft* en la wiki del colectivo[26] y presentado formalmente en el capítulo *Live Algorithm Programming and a Temporary Organisation for its Promotion*, escrito en el año 2004 por los integrantes iniciales de Toplap Adrian Ward, Julian Rohrhuber, Frederik Olofsson, Alex McLean, Dave Griffiths, Nick Collins, y Amy Alexander. El documento está enfocado en la música y expuesto, tanto en la wiki

26 https://toplap.org/wiki/ManifestoDraft.

como en el capítulo, en tres secciones. La primera sección la expongo traducida del capítulo y entre corchetes contrasto las diferencias con el texto publicado en la wiki de Toplap. Los puntos son los siguientes:

> Quienes firmamos digitalmente arriba demandamos [Demandamos][27]:
>
> - Danos acceso a la mente del/a performer, al instrumento humano en su totalidad.
> - El oscurantismo es peligroso. Muéstranos tu pantalla.
> - Los programas son instrumentos que pueden cambiar por sí mismos
> - El programa debe ser trascendido – El lenguaje [artificial][28] es el camino.
> - El código debe verse y escucharse, los algoritmos subyacentes deben ser vistos así como su resultado visual.
> - El live coding no es cuestión de herramientas. Los algoritmos son pensamientos. Las motosierras son herramientas. Es por eso que los algoritmos a veces son más difíciles de hacerse notar que las motosierras. (Ward et al., 2004; ManifestoDraft, s.f)

El manifiesto surge en medio de la controversia que causa la entrada del live coding a los circuitos musicales, desde donde se reclaman el formato de presentación y la supuesta carencia de habilidad musical relacionada con la ejecución de un instrumento. Lo anterior lo hacen notar

27 La traducción del manifiesto y todas las citas de la tesis son de mi autoría. La versión *draft* publicada en el wiki de TOPLAP, en contraste con la publicada por Ward et al. (2004), generaliza la demanda al quitar las palabras "Quienes firmamos digitalmente arriba[...]" y deja solamente la palabra "demandamos". Con este cambio en la redacción quizá dan a entender que el manifiesto pertenece a toda persona que se considere parte de la comunidad de live coding.

28 La versión ManifestoDraft de la wiki TOPLAP añade la palabra *Artificial*. Esto, posiblemente con la intención de precisar que se refieren a los lenguajes de programación.

Ward et. al (2004), quienes mencionan los textos escritos por Roger Dean y Andrew Schloss, ambos del año 2003. El primero critica el hecho de proyectar la pantalla y el segundo se queja de que los programadores aprovechan los conciertos de música por computadora para subirse a tocar a un escenario sin tener una preparación musical. Por un lado, el manifiesto responde a esta descalificación y, por otro, critica a los músicos que usan computadoras para tocar en vivo de manera "oculta", sin mostrar al público los procesos de los programas que utilizan para generar la música durante el concierto. Al respecto, Collins et al. (2003) se refieren a programas como Ableton Live y Reason a los que critican por su interfaz gráfica rígida para hacer música que contrasta con el software artesanal que estos autores, y live coders, desarrollan y utilizan. De esta manera, el manifiesto responde al momento emergente del live coding que reclama espacio para la música por computadora creada en vivo con lenguajes de programación de manufactura propia.

El primer punto del manifiesto está centrado en el humano como agente principal en la relación entre humano, lenguaje y máquina que se crea en la práctica. Alex McLean, uno de los mayores promotores del live coding, desarrollador del software TidalCycles y responsable, junto a otros live coders, de la creación de Toplap y Algorave, se posiciona a favor de una práctica centrada en el humano. En el capítulo *Computer Programming in the Creative Arts*, escrito por McLean junto a Geraint Wiggins (2012), los autores parten de la premisa de que la computación, o realización de cálculos, es una actividad que el humano ha hecho desde antes de la invención de las computadoras análogas y digitales. Tal postura se ve expresada en el manifiesto, al considerar la mente humana como el instrumento. En palabras de McLean y Wiggins el artista programador está "comprometido en una relación humana interna entre percepción, cognición y computación, y está relacionado con la notación y operación de sus algoritmos" (p. 235). Este pensamiento es reforzado por McLean en el primer capítulo del libro *The Oxford Handbook of Algorithmic Music* (2018), quien junto a Roger T. Dean dicen que la tradición del live coding no pone en primer plano la agencia del algoritmo como músico no humano, sino a "la autoría humana de los algoritmos como la actividad musical fundamental al tocar" (p. 6). El énfasis en los algoritmos como pensamiento humano, a diferencia de verlos como procesos computacionales con agencia propia, es un discurso constante en el campo del live coding.

El segundo punto del manifiesto es el que se refiere a mostrar la pantalla y quizá el más conocido. Este ha sido una especie de eslogan del live coding: “Muéstranos tu pantalla” (*Show us your screen*), antecedido por la frase “el oscurantismo es peligroso”[29]. Con este punto el colectivo Toplap demanda mostrar el código fuente a la audiencia, postura que no es exclusiva del live coding, sino que también se observa en varias prácticas de arte digital que demandan *revelar* el código detrás de la obra computacional. Por ejemplo, Hernando Barragán (2007) discute la posibilidad de que las piezas de *software-arte* provean su código para estudiarlo como material artístico y cultural. Asimismo, Steve Holmes (2016), en el contexto de la retórica del código de los videojuegos, plantea que *revelar el ocultamiento* del código en el software ayudaría a entender mejor la relación entre el humano, la tecnología y el mundo. La demanda del manifiesto por mostrar la pantalla aporta a la idea de un código computacional, artístico, abierto y expuesto al público. La acción de mostrar el código en un concierto de música por computadora es rastreada por primera vez en el ámbito del live coding, así lo menciona la sección histórica de la wiki de Toplap (s.f.) que adjudica el registro de las primeras presentaciones a Slub y a Julian Rohrhuber.

Los puntos tres, cuatro y cinco se refieren a la posibilidad de cambiar el código al vuelo mientras se programa música, a la centralidad de la expresión humana en el live coding y a lo que debe verse y escucharse al momento de proyectar código con énfasis en mostrar el código en movimiento.

Después de los postulados expuestos en la primera parte, el manifiesto continúa con una segunda sección que reconoce la continuidad de la interacción, quizá pensada desde lo que implica tocar un instrumento musical, con una preferencia por la actividad mental en la escritura de algoritmos. Los puntos que menciona son los siguientes:

> Reconocemos los continuos de interacción y profundidad, pero preferimos:
>
> - Comprensión del algoritmo

29 Esto se observa en documentales que utilizan este punto como título, tal es el caso de “Show Us Your Screens” de Louis McCallum y Davy Smith https://vimeo.com/20241649.

- La hábil improvisación del algoritmo como una [expresiva/][30]impresionante muestra de destreza mental

- Sin respaldo (minidisc, DVD, computadora de seguridad en red) (Ward et al., 2004; ManifestiDraft, s.f)

Estos puntos definen la aproximación musical de la práctica del live coding con base en la escritura de algoritmos a diferencia de la acción física ejercida sobre un instrumento musical acústico. Con ello se desplaza la habilidad física instrumental a la intelectual. La segunda sección también hace referencia a la improvisación, un punto importante en la práctica y el discurso del live coding que fundamenta su carácter exploratorio. En un inicio de la práctica del live coding hay una marcada tendencia a la práctica *from scratch* o comenzar a programar desde cero. Lo anterior se ve reflejado en la postura de no guardar el código que resulta de la presentación con lo que se hace alusión a lo efímero de la creación en el momento como sucede en la improvisación musical.

La tercera sección del manifiesto reconoce que no es necesario que la audiencia entienda el código que ve proyectado y, de manera irónica e incluso contradictoria para su discurso de inclusión, apela a una destreza al teclear código. Finalmente, el último punto de la tercera sección, que es el más extenso del manifiesto, se pronuncia sobre la interacción continua del gesto y la expresividad en la música tradicional a diferencia del proceso de escribir código para crear música, el cual no es continuo sino que hay pausas para escuchar, decidir y escribir. Los puntos de la tercera sección expresan lo siguiente:

Reconocemos que:

30 En la versión de la wiki de TOPLAP se agrega la palabra con diagonal *expressive/*.

- No es necesario que una audiencia no experta entienda el código para apreciarlo, de la misma manera que no es necesario saber tocar guitarra para apreciar una interpretación de guitarra.

- El live coding puede ir acompañado de un impresionante despliegue de destreza manual y de la glorificación de la interfaz de escritura.

- Tocar implica continuos de interacción, que abarcan tal vez el alcance de los controles con respecto al espacio de parámetros de la obra de arte, o el contenido gestual, particularmente lo directo del detalle expresivo. Mientras que las desviaciones tradicionales del tiempo de la velocidad táctil de la expresividad en la música instrumental no son aproximadas en código, ¿por qué repetir el pasado? Sin duda, la escritura de código y la expresión del pensamiento desarrollarán sus propios matices y costumbres. (Ward et al., 2004; DraftManifesto, s.f.)

Esta última sección defiende la idea de que la computadora, el software y el código que articula los algoritmos conforman un instrumento y una práctica musical que se expresa por sus propios medios, la cual llega para ocupar un lugar distinto al de la música ejecutada con instrumentos acústicos.

Algunos puntos expuestos en el manifiesto de 2004 siguen vigentes casi dos décadas después. Esto se observa en investigaciones actuales. Por ejemplo, el punto sobre mostrar la pantalla a la audiencia ha sido analizado y descrito por Shawn Lawson y Ryan Smith (2019) y Sang Won Lee (2019) o el punto sobre la cualidad cambiante del programa como instrumento musical es referido en el trabajo de Graham Wakefield y Charlie Roberts (2017).

Por otro lado, la investigación sobre el aprendizaje de máquina[31] en el live coding (Bernardo et al., 2020; Knotts y Paz, 2021; Reppel, 2020; Stewart et al. 2020; Xambó, 2020) comienza a ser relevante. Esto introduce el debate de la creatividad computacional, la agencia algorítmica no humana y empuja un desplazamiento de la centralidad del humano expresado en el primer punto del manifiesto. Tales temáticas añaden nuevas perspectivas al live coding que se integran a la extensión de las prácticas artísticas adscritas, las cuales plantean la actualización del manifiesto que se mantiene sin modificaciones por cuestiones históricas según lo expresado en discusiones sobre ello[32].

La respuesta a preservar el manifiesto en su estado original, que aún se mantiene como borrador, ha dado lugar a que la comunidad proponga otras formas de pensar e interactuar con este documento. El primer caso es el *Hacking Coreography draft manifiesto*, una versión dedicada a la danza escrita por Kate Sichhio (Toplap, 2020). Lo que hace la coreógrafa, investigadora y live coder es colocar su versión debajo del DraftManifesto. Sicchio retoma los 12 puntos originales y los reescribe en función de la mente de la coreógrafa, los movimientos y decisiones de los y las bailarinas en el escenario y la visibilidad del código. Con ello establece el papel de los algoritmos insertos dentro de una coreografía y su relación con los movimientos y gestos corporales.

Otro caso es el manifiesto feminista *livecoderA*, redactado por el colectivo de mujeres involucradas en la práctica del live coding que lleva el mismo nombre. Este manifiesto en proceso de escritura surge junto a la primera transmisión de presentaciones de live coding en línea realizadas por el colectivo el 8 de marzo de 2022. El sitio web donde está publicado el manifiesto expone en cuatro puntos “Qué, cómo y quién es una livecoderA?” (livecoderA, s.f.). El documento no retoma puntos del manifiesto original, más bien se centra en describir a la livecoderA y su relación con el código, la sociedad y la cultura. Asimismo, expone la condición de ser mujer y la necesidad de luchar por sus derechos, tanto en la vida cotidiana como en el ámbito tecnológico. Ambos manifiestos, el escrito por Sicchio y livecoderA, tratan de desbordar

31 El aprendizaje de máquina es una técnica que se refiere al entrenamiento de modelos, o conjuntos de datos, mediante el uso de algoritmos. Los conjuntos de datos obtienen la información de diferentes tipos de entradas, las cuales son analizadas por un algoritmo para dar una información de salida. Algunos modelos están basados en redes neuronales de aprendizaje profundo que alimentan sus datos a partir de descriptores.

32 Ver historial de discusiones sobre actualización manifiesto https://toplap.org/wiki/Talk:ManifestoDraft.

la convención de la práctica que resguarda el manifiesto original del live coding. Aunque no hacen explícita una crítica al manifiesto original, toman acción y no esperan más la discusión pospuesta sobre los términos en los que podría reescribirse o actualizarse el manifiesto de Toplap.

Aunado a lo anterior, el capítulo *Creativity and the Social in Algorithmic Music* escrito por Christopher Haworth (en Dean y McLean, 2018) realiza un análisis del manifiesto a nivel discursivo basándose en la teoría del actor red de Bruno Latour. De manera general, Haworth encuentra una postura que trata de desarticular a la música por computadora a través de lo social. Considera que el live coding más que un estilo musical es una forma de práctica, socialización y técnica, que pretende revelar la música por computadora en su estado elemental a la que llama "música por computadora auténtica". Lo anterior sucede con el desmarque que hace el live coding de las prácticas de música por computadora de interfaz fija como Ableton Live a través del uso del lenguaje de programación. La idea sobre la música por computadora auténtica atribuida por Haworth al live coding parte, dice el autor, de un concepto mutable que enmarca dispositivos técnicos, expectativas sociales, política y ontología de lo abierto-terminado. Así, dice Haworth, el Manifiesto expresa una retórica de la música por computadora auténtica en vivo propuesta por Toplap. Resalta el énfasis que hace el manifiesto de la música como idea abstracta por encima de las herramientas y de la transparencia de una música idiomática que sucede en vivo para el humano y en tiempo real para la computadora.

El manifiesto de Toplap, a pesar de que ha dejado de responder a muchas de las formas del live coding actual, continúa siendo un texto atractivo para quien se adhiere por vez primera a esta práctica. Esto quizá se debe a la forma de plantear la música con computadoras y algoritmos que apela a lo lúdico de la cultura hacker más que al estudio de la música. Este documento responde a las características que Paz Sastre (2021) atribuye a los manifiestos artísticos publicados en red durante los años 90 en diferentes medios digitales e impresos. En su libro, *Manifiestos sobre el arte y la red 1990-1999* (2021), Sastre propone que los manifiestos de las culturas en red expresan bastante "el papel de las artes y las tecnologías en las sociedades globalizadas" (p. 15). Asimismo, la autora otorga al manifiesto en la red una capacidad de alcanzar en poco tiempo la imaginación y la acción de una gran audiencia. Esto se ve reflejado

en lo que ha logrado el manifiesto de live coding de Toplap al ser publicado en internet siguiendo la estrategia de comunicación de las prácticas y el activismo de la década que le antecede.

Hoy, el live coding es una práctica establecida que sigue creciendo, es incluida en más circuitos artísticos y académicos y cada vez más gente participa en actividades organizadas por varios de sus actores. El circuito y las redes alrededor del live coding permiten al principiante una rápida incorporación a eventos como los organizados en torno al movimiento Algorave, que junto a distintas prácticas de live coding han creado sus espacios y formas de expresión. Esto se ve reflejado en la series de simposios sobre live coding, conciertos presenciales y en red, así como en la extensión de la comunidad inicial, centrada en Reino Unido y Alemania, que se ha diversificado en circuitos locales de otros países.

En este sentido, el live coding tiene en Toplap una instancia que permite a cualquiera declararse participante de esta práctica global. La adscripción se realiza cuando una iniciativa colectiva o individual se constituye para promover la práctica de live coding de manera local y se declara *nodo* de Toplap para dar presencia al colectivo en una región. Lo anterior se observa en un incremento de nodos que también incluyen representantes del movimiento Algorave, colectivos de live coding y grupos universitarios. Algunos nodos organizan eventos de live coding de manera activa, crean sitios web con contenidos locales o páginas en Facebook, diseñan su logo y en algunos casos desarrollan un discurso propio como se observa en los casos de CLiC (Colectivo de Live Coders), NL_CL (Netherlands Coding Live), Livecode.NYC, Tacacocodin, Comunidad de Live Coders Perú y ToplapMx. Otros nodos simplemente traducen o repiten la descripción del sitio web de Toplap y en algunos casos los sitios web donde se hospedan están desactualizados o han caducado[33].

A su vez, se pueden mencionar los casos de escenas locales ligadas a una institución, universidad o espacio independiente que simpatizan con el live coding definido por Toplap pero que no necesariamente se han declarado como un nodo del colectivo. Estos casos han creado un imaginario propio de la práctica como el caso de la escena de live coding del Centro Multimedia del CENART en la Ciudad de México, la comunidad de investigación del laboratorio NIL de la

33 Aquí se puede ver la diversidad de nodos que representan a Toplap en diferentes países: https://toplap.org/nodes/.

Universidad McMaster en Hamilton, así como el proyecto de difusión de live coding europeo On-the-Fly.

Asimismo, el live coding se ha visto potenciado por el movimiento Algorave iniciado en el año 2011 por Alex McLean y Nick Collins. Dicho término es una palabra compuesta que hace alusión a las palabras algoritmo y *rave*, esta última hace referencia a un término que pasó de denominar las fiestas de *acid house* de finales de los años 1980 a ser un sinónimo de "salir de fiesta" (Shulman, 1999, p. 200). El movimiento Algorave desplaza el live coding de corte experimental en formato de concierto a la fiesta de música electrónica en el *club* y el *pub* inglés. Más tarde, el movimiento se ha extendido a géneros musicales locales, por ejemplo, la exploración de la cumbia sonidera en la Ciudad de México descrita por el colectivo RGGTRN (Ocelotl et al., 2018).

Por otro lado, aunque la comunidad del live coding promueve los valores de descentralización, inclusión, diversidad y horizontalidad y mantienen una invitación abierta a pertenecer a esta práctica, no se puede dejar de mencionar que ello también conlleva contradicciones y exclusiones. Si bien la idea de los nodos promueve la inclusión de comunidades locales, en muchos casos, esto ocurre bajo el modelo y discurso de Toplap. El intento por descentralizar y crear una comunidad global horizontal, a través de lo local, establece una jerarquía por regiones donde la geografía, la infraestructura de movilidad y la economía facilitan la conexión entre nodos, sobre todo alrededor del centro de Europa. Mientras que, en el caso de América la extensión geográfica y la falta de recursos, sobre todo en la región latinoamericana, genera un aislamiento que impide generar circuitos más allá de lo virtual. Por otro lado, el live coding pugna por un saber hacer abierto en el ámbito artístico aunque parece que los temas de desarrollo tecnológico siguen reservados para los expertos. A todo esto se suman exclusiones que, si bien no son exclusivas del live coding, forman parte del problema que plantea el acceso a las tecnologías computacionales necesarias para el arte como aquellas de índole económica, lengua, género y procedencia.

Por último, el live coding es un campo en transición entre lo emergente y lo establecido, proceso en el que se continúa discutiendo y definiendo sus alcances tecnológicos, artísticos, científicos y sociales. Ciertamente el manifiesto de Toplap es un documento de referencia muy

importante para entender el live coding, sin embargo, dista mucho de representar e incluir lo que hoy es esta práctica. Respecto a esto, distintos colectivos y personas del ámbito han optado por hacer sus manifiestos en torno al live coding en lugar de actualizar el documento original. Esto habla de la necesidad de seguir reflexionando sobre qué es el live coding y del manifiesto como estrategia para hacer circular las ideas que definen a una práctica artística desde diferentes perspectivas.

## 2.4 Desarrollo tecnológico basado en el live coding

El inicio del live coding empujó el desarrollo de lenguajes de programación personalizados y objetos computacionales junto a la práctica artística. No obstante, con el paso del tiempo la práctica artística y el desarrollo tecnológico se han ido separando debido a una especialización de cada ámbito y a la tendencia del live coding a estandarizar el uso de lenguajes de programación establecidos. El inicio que partió del intercambio y la colaboración entre artistas, programadores y tecnólogos con inclinación al software artesanal, más tarde marcó una división entre desarrolladores de lenguajes de programación y grupos de usuarios de esos lenguajes. Lo anterior sucedió bajo el discurso de la libertad de ejecución del software y la apertura del código fuente de los programas con fines de estudio y posible modificación, principios establecidos por el software libre. A pesar de esta división, el desarrollo de lenguajes de programación y objetos computacionales para el live coding no ha dejado de ser una actividad recurrente como lo muestran los siguientes ejemplos.

Actualmente hay diversos desarrollos tecnológicos en el live coding en los que se puede observar una relación con la práctica artística: metalenguajes de programación, interfaces, librerías, plataformas y sistemas como en los casos de Sema, CaosBox, INSTRUMENT, Hydra y MIRLC. Estos ejemplos tienen una aproximación personal a la práctica del live coding de sus autores y autoras. Ello se ve reflejado en las perspectivas que los caracterizan como el diseño de lenguajes, el diseño de interfaces, la retroalimentación de video y la recuperación de información musical. Escogí estos desarrollos porque han influenciado la forma en la que me aproximo a la

práctica del live coding y porque en algunos casos he utilizado estas tecnologías para probar mis ideas de desarrollo tecnológico. Además, conozco personalmente a quienes los desarrollan, lo que me permitió referirme a sus trabajos desde tutoriales, repositorios y artículos publicados, así como desde sus propias descripciones.

El primer ejemplo es Sema[34], una plataforma desarrollada por Thor Magnusson, Chris Kiefer y Franciso Bernado bajo el proyecto de investigación *Musically Intelligent Machines Interacting Creatively* (MIMIC)[35]. Sema, desarrollada en JavaScript, sirve para hacer live coding musical, diseñar lenguajes pequeños de live coding y entrenar modelos de aprendizaje de máquina. No es necesario instalarla pues funciona en los navegadores Chrome y Chromium[36]. La plataforma muestra diversos niveles de abstracción en sus ventanas que dejan ver varios procesos mientras programamos. Por ejemplo, la ventana de escritura de código y las ventanas *debuggers*, estas últimas incluyen un editor de gramática donde se escribe el lenguaje de programación o el árbol de sintaxis abstracta que nos muestra la estructura del lenguaje cuando evaluamos una línea de código.

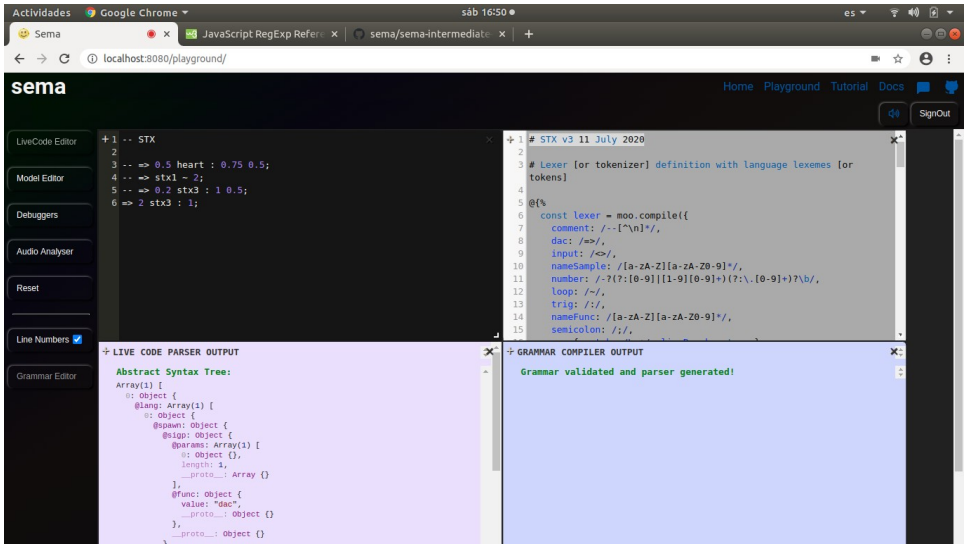



*Figura 2.1. Plataforma Sema con el mini lenguaje de live coding STX. Captura de pantalla.*

34 Se puede consultar su código fuente y su documentación en https://github.com/mimic-sussex/sema.
35 https://mimicproject.com/.
36 La plataforma Sema funciona en Chrome o Chromium en siguiente URL https://sema.codes/.

Las consecuencias tecnológicas de usar esta plataforma se dan al combinar un entorno para usuarios programadores y desarrolladores. Sema tiene un lenguaje de programación sencillo para hacer live coding que está diseñado para mostrar cómo funcionan diferentes procesos computacionales mientras programamos. Sus desarrolladores están conscientes de la estandarización de la práctica de live coding generada por el uso de los lenguajes de programación establecidos. Por ello, uno de los objetivos de Sema es fomentar el diseño de lenguajes de programación para el live coding y detonar nuevas ideas en la expresión musical del campo. En este sentido, proponen romper la barrera que supone la programación como forma de expresión y la programación para el desarrollo de herramientas tecnológicas. En este caso, un lenguaje de programación puede ser usado al mismo tiempo que es desarrollado, de esta manera la práctica artística informa al desarrollo tecnológico, y viceversa, en el mismo momento de programarlo y programar con él.

El segundo ejemplo es CaosBox[37], un secuenciador de interfaz gráfica y código desarrollado por José Carlos Hasbun con el lenguaje de programación SuperCollider. CaosBox puede secuenciar y procesar sonidos sintetizados o grabados mediante las funciones de la interfaz gráfica o escribiendo código. Este secuenciador fue desarrollado en el contexto de la escena mexicana de live coding, en la que algunos de sus actores han comenzado a crear software, lo que revela una práctica artística que pasa de apropiarse de herramientas tecnológicas a crearlas. En este caso, el diseño de software personal es importante pues responde a una forma particular de aproximarse a la práctica creativa. Es decir, se dejan de utilizar las soluciones propuestas por otros desarrolladores para implementar las propias y, con base en ellas, crear música. El código fuente de CaosBox se encuentra en un repositorio de GitHub, por lo que es posible descargarlo para instalarlo y utilizarlo como extensión de SuperCollider. Su instalación requiere conocer el programa sobre el cual está desarrollado, lo cual se especifica en las instrucciones escritas en el documento README del repositorio. El diseño de la interfaz propone dos niveles de uso: uno guiado por rejillas, botones y deslizadores propios de una interfaz gráfica de instrumentos musicales digitales y otro por el código fuente. El primero no requiere programar, solo explorar la interfaz presionando o deslizando sus componentes, mientras que el segundo requiere escribir código fuente para complementar y modificar los

37 https://github.com/josecaos/caosbox.

valores de la interfaz. La interfaz gráfica coloca el secuenciador en un nivel de abstracción más alto que el de la programación, lo que permite producir y modificar sonido de manera inmediata en contraste con la escritura de código fuente.

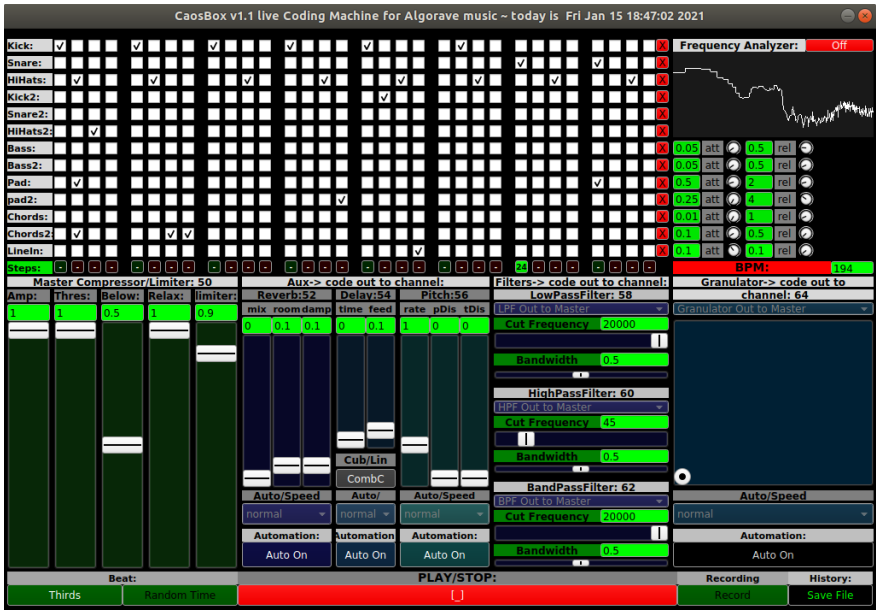


*Figura 2.2. Interfaz gráfica de CaosBox. Captura de pantalla.*

El tercer ejemplo es INSTRUMENT[38], una librería para hacer live coding desarrollada por Rodrigo Frenk en SuperCollider. Esta librería provee recursos musicales como ritmo, armonía y melodía, junto a procesamiento de sonido. Al igual que el ejemplo anterior, surge en el contexto de la escena mexicana de live coding. La mezcla de exploración de la práctica artística y la experiencia en el desarrollo de software incentivan este tipo de aproximaciones en las que cada desarrollador añade sus soluciones. Posteriormente, cuando se abre la posibilidad de uso a otros usuarios, estos negocian con una forma de pensamiento inserta en el software que deben entender e incorporar a su propia práctica artística.

INSTRUMENT es definida por su desarrollador como una librería puesto que es desarrollada dentro del ecosistema de SuperCollider, aunque por la sintaxis resultante tiene rasgos de un lenguaje de programación musical. En el documento de introducción README del

38 https://github.com/punksnotdev/INSTRUMENT.

repositorio, Frenk invita a usar la librería y advierte sobre su desarrollo en proceso y las posibles fallas debidas a su versión pre-alpha. Esto se ve reflejado en los constantes *commits* o cambios que sufre el repositorio de la librería[39], por lo que continuamente queda desactualizada. Debido a ello, cuando corremos los nuevos ejemplos que presenta el documento README surgen errores a menos que se realice un *pull* o actualización del repositorio en la computadora que tiene instalada la librería. Una característica de este desarrollo, que la relaciona con su contexto, es la sugerencia de usarla para realizar live coding *from scratch* o partiendo desde cero. Según Villaseñor y Paz (2020), esta modalidad es una forma de realizar live coding que caracterizó a la comunidad de la Ciudad de México dentro del Cmm, en la que tanto Frenk como Hasbun estuvieron bastante activos.

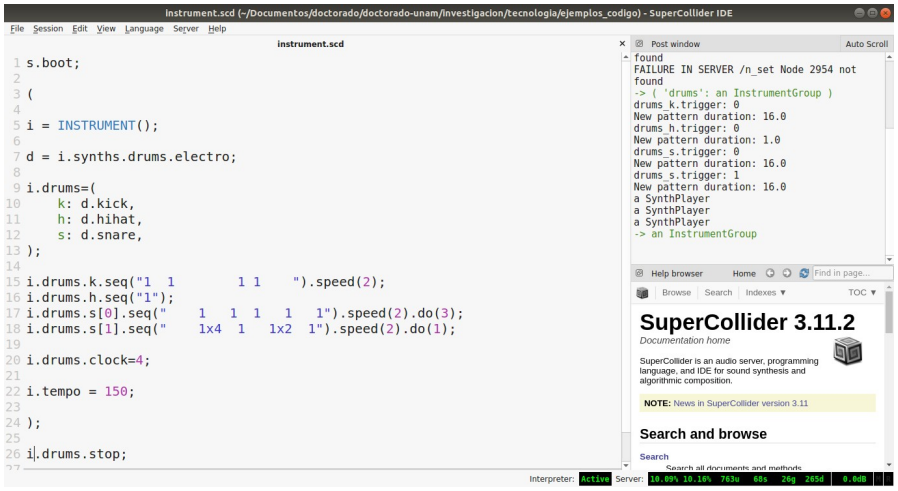



*Figura 2.3. Extracto de código escrito en INSTRUMENT por punksnotdev. Captura de pantalla.*

El cuarto ejemplo es el lenguaje de programación visual Hydra[40] desarrollado por Olivia Jack alrededor de 2018 en JavaScript para ser usado en un navegador. Este desarrollo parte de

39 Las palabras *commit*, *README.md* y *pull* forman parte del vocabulario de los comandos para realizar actualizaciones al código fuente de los repositorios Git. *Commit* se refiere a un comentario sobre un cambio realizado que, a partir de un identificador, permite rastrearlo en la historia de cambios de un proyecto. *README.md* es el documento inicial del repositorio en el que generalmente se hace una introducción del proyecto y el comando *pull* permite actualizar un repositorio local respecto a los cambios más recientes del repositorio principal.

40 https://github.com/ojack/hydra.

varias ideas: la retroalimentación de video, la síntesis modular y facilitar la enseñanza de código creativo sin instalar software. Su desarrollo comienza desde un planteamiento estético, pedagógico y de acceso al live coding visual. Hydra es un software bastante utilizado en los circuitos del live coding, sobre todo aquellos relacionados con el movimiento Algorave. Al igual que los anteriores ejemplos, la desarrolladora inserta la búsqueda de una estética personal a las posibilidades tecnológicas del programa. Este proyecto se caracteriza por la comunidad activa a su alrededor, lo que se ve reflejado en la cantidad de ejemplos programados por los usuarios incluidos de manera aleatoria al iniciar Hydra dentro de un navegador. Lo anterior se refuerza con los encuentros regulares que ocurren desde el año 2020 llamados *Hydra meetups*[41], en los que diversos integrantes de la comunidad de usuarios exponen sus proyectos. Cabe mencionar que presento un ejemplo de desarrollo dedicado a la imagen por la características que comparte con lenguajes de programación de sonidos como la lógica de programación, la relación con la síntesis modular y las posibilidades para acoplarse a proyectos sonoros. Estas características me permitieron no solo estudiar el lenguaje de programación sino incorporarlo a mi flujo de trabajo para generar proyectos audiovisuales.

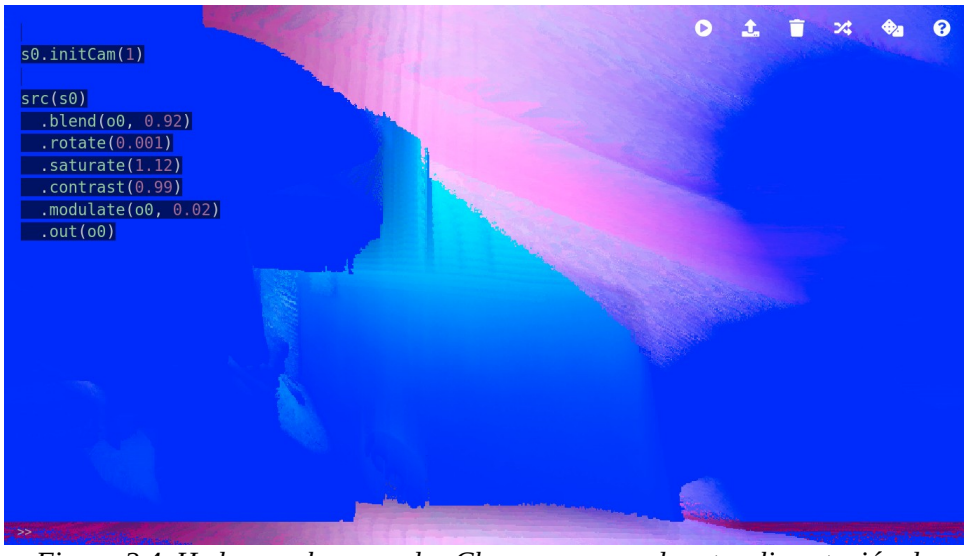



*Figura 2.4. Hydra en el navegador Chrome generando retroalimentación de video producida con una cámara web conectada a la computadora. Captura de pantalla.*

41 https://hydra-meetup-3.glitch.me/.

Como último ejemplo presento MIR Live coding o MIRLC[42], un desarrollo que más adelante se transformó en el proyecto *Agente Virtual para la Recuperación de Información Musical en Live Coding* o MIRLCa[43]. Ambas extensiones de SuperCollider son desarrolladas por Anna Xambó desde 2016 y 2020 respectivamente. Estas consisten en varias *clases* diseñadas para utilizar técnicas MIR y, en el caso de MIRLCa, de aprendizaje de máquina en el live coding.

Respecto a MIRLC, Xambó et al. (2019) mencionan que MIR (M*usical Information Retrieval*) se refiere al análisis y recuperación de información musical contenida en grabaciones de sonido y que las técnicas de extracción de información musical pueden usarse en el live coding para obtener información del timbre y contenido de la improvisación, así como para recuperar archivos de bases de datos personales o *crowdsourced* como Freesound. Este trabajo intenta poner en contacto a las comunidades del live coding y MIR.

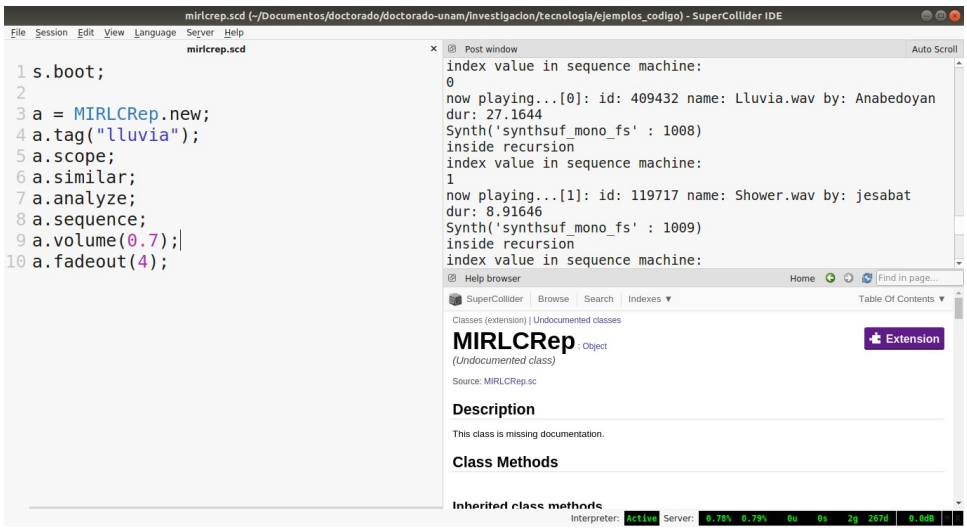



*Figura 2.5. MIRLCRep, sonidos por Anabedoyan, id: 409432 y jesabat, id: 119717. Captura de pantalla.*

MIRLC es una investigación que despliega desarrollo tecnológico, investigación teórica y práctica artística. Como menciona la autora y desarrolladora, ella junto a sus colaboradores han

42 https://github.com/axambo/mirlc.
43 https://mirlca.dmu.ac.uk/.

revisado el estado del arte de las técnicas MIR en el live coding y pretenden que tal desarrollo tecnológico tenga una salida artística. Esto implica el acercamiento del live coding al uso de bases de datos sonoros complejas, a diferencia de los materiales de sonido con los que trabajan los live coders que vienen incluidos en el software o son grabados por ellos mismos. Por el contrario, MIRLC nos enfrenta a recuperar sonidos entre miles de grabaciones desconocidas almacenadas en bases de datos públicas. Lo anterior conlleva el problema de utilizar técnicas MIR, pues estas demandan mucho proceso computacional y no son óptimas para funcionar en tiempo real, contrario al live coding que se caracteriza por una respuesta inmediata y poco proceso computacional. Por último, MIRLCa añade técnicas de aprendizaje de máquina con base en el gusto musical del live coder, quien entrena a un agente virtual para navegar la base de datos de Freesound y seleccionar sonidos.

Los ejemplos anteriores muestran desarrollos tecnológicos con distintos niveles de abstracción en el live coding, los cuales exponen desde el funcionamiento completo de un lenguaje de programación hasta una pieza de software que forma parte de estos. En un acercamiento comparativo podemos observar que la interfaz de la plataforma Sema muestra varios procesos desplegados en sus ventanas que dejan ver diferentes niveles de procesamiento en los que actúa el código fuente mientras se modifica el programa. Por su parte, CaosBox permite decidir cómo operar el programa, ya sea con su interfaz gráfica o escribiendo código. En este caso la interfaz gráfica permite construir y procesar rápido la música, mientras que la escritura de código resulta más lenta y sus cambios no son tan evidentes en el momento de aplicarlos. Hydra es un ejemplo de la integración del código al resultado visual expuesto con un lenguaje declarativo de manera clara. Las funciones de Hydra se interconectan para modificar la imagen que genera el software, el cual parte de una búsqueda estética particular de su desarrolladora; esto determina en gran medida el resultado visual a diferencia de otros lenguajes que parten de gráficos primitivos. Por otro lado, la programación compleja de las clases MIRLC y MIRLCa permanece oculta al usuario al momento de tocar, contrario a su interfaz de código compuesta por mensajes simples que permiten crear música con sonidos recuperados de la plataforma Freesound.

Muchas veces estos proyectos comienzan como una herramienta personal, pero con el tiempo, y gracias a la dinámica del código abierto, resultan del interés público y se convierten en programas muy usados: se pasa de un desarrollo tecnológico personal a uno de uso comunitario que construye y define su práctica con dicha tecnología. En este sentido, el software personal se expande a través de una comunidad que se apropia de la herramienta y la utiliza en distintos contextos culturales, aunque a veces esto implica una estandarización de la práctica. Esto rara vez sucede de manera espontánea, la extensión de su uso va acompañada del esfuerzo de enseñanza y difusión de quienes desarrollan software, de crear una comunidad y mantenerla activa. Una forma de llevar a cabo tal difusión se da principalmente por la impartición de talleres, así como por la organización y gestión de foros, encuentros y ciclos de conciertos. Lo anterior se puede observar en las actividades de difusión de la plataforma Hydra que mantiene reuniones mensuales llamadas *hydra-meetup* en las que participan miembros de su creciente comunidad, la serie de talleres de Sema y de MIRLCa que culminan en conciertos y entrevistas o el taller de INSTRUMENT durante el 1er Encuentro Transferencias Aurales, en el que Frenk creó la comunidad de este desarrollo e invitó a unirse a quienes tomaban el taller[44].

Los ejemplos discutidos en esta sección ocurren en contextos dentro y fuera de la academia, esto implica que circulan en distintos circuitos del live coding. No obstante, tienen en común que sus desarrolladores y desarrolladoras los colocan en repositorios de la plataforma GitHub, la cual, como sugiere la página *about* de su sitio web, contempla un espectro que va del desarrollador independiente hasta la compañía entera. Al colocar el código fuente de un desarrollo en GitHub este ingresa a una dinámica de código abierto u *open source* y establece una conexión con el software libre a través de sus licencias.

Si bien, GitHub es una plataforma de desarrollo de software que declara su dinámica de trabajo como código abierto, la conexión que establece con el software libre se puede apreciar en el tipo de licencias que permite agregar a sus repositorios como *General Public License* o GPL,

44 Los encuentros de Hydra se llevan a cabo desde agosto de 2020 https://hydra-meetup-0.glitch.me/. El taller de Sema del verano 2020 culminó con la presentación de proyectos realizado en el marco del Network Music Festival en la sesión de NMF x MIMIC new works concert https://networkmusicfestival.org/. La serie de talleres sobre MIRLC en su nueva versión MIRLCa fue realizada entre finales de 2020 e inicios de 2021, la cual culminó con una serie de conciertos y entrevistas https://mirlca.dmu.ac.uk/. Por último, el registro del taller de INSTRUMENT impartido durante transferencias Aurales https://youtu.be/PL_MpfwB-u8.

AGLP y MIT. Asimismo, la opción de que los repositorios sean públicos permite que cualquiera pueda accesar a sus contenidos. Esto conduce la intención del desarrollo de software para el live coding a un pensamiento más afín al software libre que se enfoca en los derechos y las libertades del individuo para usar software más que en hacer eficiente su proceso de desarrollo. De esta manera, los ejemplos expuestos en esta sección entran en la misma arena de exposición, independientemente de sus afiliaciones, lo que permite el acceso a su código fuente para instalarlo, modificarlo, estudiarlo y utilizarlo, así como al conocimiento que se deriva de ello.

Asimismo, aunque GitHub es una plataforma orientada a un modelo de negocios y no tanto a la investigación científica, la educación o el arte, sus repositorios se vuelven contenedores de proyectos y fuentes de consulta en esos ámbitos. En este sentido, la intención de utilizar repositorios para el código fuente en la investigación académica se encuentra conectada con la postura de la *ciencia abierta*.

En cuanto al tema de la descarga de software libre y *open source* para su uso, una característica que diferencia a Sema e Hydra de los otros ejemplos es que funcionan en un navegador. Esto evita descargar e instalar software en la computadora, lo que en algunos proyectos experimentales resulta complicado y desmotiva a los usuarios menos expertos. Si bien el hecho de colocar el software en un servidor para accesar desde un navegador tiene ventajas, hay otro tipo de problemas a los que se enfrenta. Por ejemplo, en el foro de Toplap, James Harkins (2020), profesor del Conservatorio de Música de Xinghai en Guangzhou, expresa la dificultad para instalar un servidor en China para correr aplicaciones externas, así como el filtro de algunos recursos de los navegadores web que permiten funcionar plataformas en red como Estuary o Troop.

Por otro lado, la instalación de programas y extensiones contenida en repositorios, como el caso de CaosBox, INSTRUMENT y MIRLC, requiere saber cómo descargar o clonar el código de los repositorios y, una vez descargados, cómo instalar tales extensiones. A veces, esto requiere modificar el código original y entender los errores que surgen durante la instalación. Sin un conocimiento previo sobre los procedimientos de instalación se crea una barrera para accesar y usar estos desarrollos. Otra situación que sucede constantemente con software en proceso de desarrollo es la diferencia de versiones de los recursos que un software de uso libre requiere para

funcionar. Esto se observa cuando el desarrollo se hace pensando en el hardware más actual, situación que lo hace incompatible con equipos que trabajan con versiones de sistemas operativos menos actuales y hardware más viejo. También puede suceder de forma contraria, al adquirir una nueva computadora con un sistema operativo actualizado pueden dejar de funcionar algunos componentes hardware como las tarjetas de sonido externas y desarrollos que no reciben mantenimiento. En este contexto, la intención de los desarrolladores y desarrolladoras es mantener el código fuente accesible, sin embargo, hay una demanda de conocimiento especializado para participar en dinámicas que van más allá de instalar software y utilizarlo. La barrera del software que separa la práctica artística del desarrollo tecnológico comienza a desdibujarse al tener acceso a estas tecnologías. No obstante, en el momento de contribuir sin el conocimiento tecnológico especializado, el idioma de la documentación y los medios necesarios, esa barrera parece infranqueable, de lo que se desprende la pregunta ¿qué tan abierto es lo abierto?

## 2.5 El estudio del live coding

La palabra live coding muestra su relación con la programación en la palabra *coding* o codificación, la cual en las ciencias de la computación se refiere al proceso de implementar una solución computacional diseñada en forma de algoritmo. Para que las soluciones algorítmicas planteadas sean operables dentro de un programa de cómputo estas deben ser escritas como código fuente utilizando algún lenguaje de programación. El live coding, a su vez, está vinculado con diversos campos de estudio que van de las ciencias de la computación, la tecnología musical hasta las humanidades digitales. Asimismo, el live coding opera sobre diversos desarrollos tecnológicos, principalmente lenguajes de programación como SuperCollider, Tidal Cycles, Sonic Pi o Hydra. Esto implica una doble aproximación, artística y tecnológica, que puede suceder de forma separada o conjunta. Sin embargo, estudiar el live coding solo desde tales perspectivas reduce su estudio a sus aspectos estéticos y tecnológicos dejando de lado otras

cuestiones de tipo social que resultan importantes para la forma comunitaria en la que está configurada esta práctica.

Desde su comienzo, el live coding ha sido estudiado por los campos de la música y las ciencias de la computación y, posteriormente, desde el arte y las humanidades. Con ello, los enfoques de estudio del live coding se han ampliado más allá de la relación artística y tecnológica. Como sucede en muchos campos recién formados, una de las primeras preguntas que surgió fue la que trató de dar una definición de la práctica. Tal pregunta ha sido contestada y reformulada desde varias perspectivas; primero, desde las posibilidades tecnológicas de los lenguajes de programación y la escritura de algoritmos al vuelo y, segundo, desde diferentes campos de estudio como la sociología, la antropología, la cognición o las comunicaciones.

Una de las definiciones más citadas, y de las primeras en ser publicadas, es la formulada por Adrian Ward, Julian Rohrhuber, Frederik Olofsson, Alex McLean, Dave Griffiths, Nick Collins y Amy Alexander (2004), quienes propusieron que el "[l]ive coding es la actividad de escribir (partes de) un programa mientras este corre" (p. 243). En esta cita el live coding es definido de manera funcional como una actividad de programación. Más tarde, el propio McLean (2011) definió el live coding en términos de una práctica artística e investigativa como "la actividad de un grupo de practicantes e investigadores quienes han comenzado a desarrollar nuevas aproximaciones para hacer música por computadora y video animación" (p. 129).

Además del término live coding, otros términos han sido utilizados para definir la práctica de codificar, o programar, en vivo con fines creativos. McLean (2011) menciona las expresiones *interactive programming*, *on-the-fly programming*, *conversational programming*, *just-in-time programming* o *with-time programming* a las que se suma *live programming* (HFBK, 2004). McLean comenta que si bien tales expresiones tratan de definir lo mismo, el término live coding se distingue por el énfasis que hace en la cualidad improvisatoria de la práctica.

El origen de algunos de los términos anteriores se pueden rastrear fácilmente. Por ejemplo, es bien conocido que el término *on-the-fly programming* o *programación al vuelo* fue acuñado por Ge Wang y Perry Cook en el año 2004, el que describieron de manera similar a la definición de live coding propuesta por Ward et al. (2004). Wang y Cook plantean que "*On-the-*

*fly programming* es un estilo de programar en el que el programador/intérprete/compositor aumenta y modifica el programa mientras está corriendo, sin detenerlo o reiniciarlo, para afirmar un control expresivo y programable en tiempo de ejecución" (2004, p. 138).

Ambas definiciones hacen énfasis en la posibilidad de mantener el programa corriendo, una característica técnica que dio lugar a la creación de música por computadora en el momento que permite improvisar con un lenguaje de programación. Esta característica fue aprovechada a inicios de los años 2000 por algunos programadores-artistas, como los integrantes de slub (Alex McLean y Adrian Ward), Julian Rohrhuber y Ge Wang, quienes crearon los primeros lenguajes de programación y librerías musicales para crear música con programación al vuelo. Tales piezas tecnológicas son aplicaciones para crear música con los lenguajes de programación Perl y REALbasic utilizadas por slub (Coollins et al., 2003), la librería JITLib de SuperCollider creada por Julian Rohrhuber (Collins et al., 2003) y el lenguaje de programación musical ChucK desarrollado por Ge Wang y Perry Cook (Wang y Cook, 2004).

Además de la posibilidad tecnológica mencionada en el párrafo anterior, lo que ayudó a conformar el live coding de manera conceptual fue: 1) la inclinación por la creación artística con lenguajes de programación, 2) la demanda de hacer visibles la escritura de código y los algoritmos empleados durante las presentaciones de música por computadora, 3) la posibilidad que brinda un programa intérprete para modificar el código fuente de un programa instrucción por instrucción al mismo tiempo que opera, 4) el entendimiento de la computadora y su lenguaje de programación como partes que conforman un instrumento musical digital y 5) la legibilidad y facilidad de escritura de los lenguajes de programación de alto nivel.

De manera posterior a las definiciones fundacionales antes descritas, diferentes aportes han expandido el término en función de lo social, tal es el caso de Carolina Di Próspero (2015, 2017, 2019) quien toma al live coding como objeto de estudio de la antropología. En sus artículos, Di Próspero se refiere al live coding como una expresión musical digital colectiva que enfatiza un híbrido compuesto por lenguaje de programación y artista en un contexto de improvisación y exploración. Para Di Próspero es importante observar el cuerpo en el circuito de retroalimentación que se forma en la programación artesanal donde se construye una relación con la tecnología a partir de un diálogo entre programador y programa. El ciclo de

retroalimentación en el live coding al que se refiere Di Próspero es un proceso que han descrito previamente Alex McLean (2011) y Marije Baalman (2015).

McLean (2011) propone que el ciclo de retroalimentación del live coding está basado en la programación *bricolage* que involucra la imaginación, la codificación, la escucha y la decisión. Por su lado, Baalman (2015), quien es programadora-investigadora, artista computacional y live coder, propone un flujo por el que pasa el código desde que es concebido en el pensamiento hasta su paso por diferentes componentes de la computadora y su posterior reingreso a la mente humana. La autora menciona que en este ciclo, tanto humano como computadora, *in-corporan el código* a sus cuerpos[45].

Todo lo anterior se refleja en el crecimiento de las temáticas del live coding, las que se pueden seguir desde los congresos que tratan el tema de forma especializada o que han incluido el tópico de forma recurrente. Por ejemplo, las conferencias International Conference on Live Coding (ICLC) y New Interfaces for Musical Expression (NIME), desde donde se pueden rastrear los siguientes temas: desarrollo de lenguajes de programación, música colaborativa, in-corporación (*embodiment*) del código, gestualidad, inteligencia artificial, visualización del código, live coding junto a instrumentistas, etnografía del live coding, antropología del live coding, discapacidad, laptop como instrumento, bordado, cognición, escucha de máquinas, composición e improvisación, música en red, live coding visual y audiovisual, ontología y epistemología del live coding, live coding en navegadores web, sociología, entornos participativos, danza, estudios de caso, recuperación de información musical, diseño de instrumentos musicales, sonificación de datos, interfaz, live coding de hardware, live coding de bajo nivel, análisis de la forma musical, control de luces y live coding durante la pandemia.

En el caso de Latinoamérica hay un marcado interés por las temáticas del hackfeminismo, el ciberfeminismo, lo decolonial y el cómputo ancestral aplicado al live coding. Las discusiones en torno a los temas anteriores ocurren fuera de conferencias establecidas, o en vías de establecerse, y generalmente privilegian la oralidad del conversatorio frente a la escritura del *paper* académico. Ejemplos de lo anterior son el Festival Pumpumyachkan organizado por la

45 Nota de la traducción: Marije Baalman utiliza el término en inglés “embodiment of code”.

plataforma Asimtria[46] o el Encuentro Transferencias Aurales organizado por el Seminario Permanente de Tecnología Musical del Posgrado en Música de la UNAM[47]. En este sentido, los registros de tales conversatorios y exposiciones se pueden recuperar de las plataformas YouTube, Facebook live o Wordpress. Ahí se encuentran registradas videoconferencias realizadas con Zoom o Jitsi que fueron potenciadas por la comunicación a distancia durante la pandemia. De los casos mencionados, Transferencias Aurales es un ejemplo de la búsqueda de formatos alternativos a las memorias de congreso. En este caso, se hicieron unas memorias colectivas llamadas Mnemozine que, a diferencia del formato convencional, se realizaron como un ejercicio colaborativo y comunitario desde la metáfora de abrir el código de un encuentro artístico y académico[48].

De las temáticas mencionadas en los párrafos anteriores, aquella que trata la capacidad del live coding para generar conocimiento ha sido poco expuesta. Uno de los escasos artículos, o quizá el único, que problematiza el tema es *What does live coding know?* escrito por Geoff Cox (2015), quien expone la condición onto-epistemológica del live coding en el contexto de la investigación artística y su potencial crítico para generar nuevas formas de conocimiento con la programación. El autor menciona que la incertidumbre, como valor implícito de la práctica de live coding, puede constituir una metodología que desestabiliza el conocimiento desde las formas emergentes que produce la improvisación. Para ello, Cox dice que es necesario entender qué es el código, cómo se produce y en qué se convierte. Asimismo, plantea que la investigación artística puede ser un terreno para lidiar con las diversas formas de conocimiento del live coding.

Como hemos visto hasta aquí, el live coding es una práctica artística computacional que ha sido estudiada desde varios campos. Este proceso involucra el estudio de sus aspectos desde diferentes perspectivas al mismo tiempo que emergen nuevos modos de hacer live coding. Para mí, la perspectiva de investigación artística es una vía que puede ser aprovechada para construir conocimiento a partir de los procesos del live coding.

Según Borgdorff (2012), la investigación artística, o investigación en el arte, es un campo emergente donde hay un encuentro del arte y la academia y en el que se entretejen la práctica y la

46 Pumpumyachkan: https://asimtria.org/pumpum/.
47 Transferencias Aurales: https://limmefamus.wordpress.com/transferencias-aurales/.
48 Mnemozine: https://limmefamus.wordpress.com/mnemozine/.

teoría. En este terreno, dice, hay una contribución de las prácticas artísticas a la investigación y se da una apertura por parte de la academia hacia las formas de conocimiento que las prácticas artísticas generan. Por su parte, López-Cano y San Cristóbal (2014) concuerdan que la investigación artística es un campo nuevo de investigación y creación donde confluye pensamiento artístico y académico-científico. Para estos autores la investigación artística demanda un perfil del artista que se interese por la investigación, la reflexión sobre la práctica y la problematización de su actividad artística. Específicamente, dicen que, en el ámbito de la música las preguntas de investigación están vinculadas a la creación y se dirigen a la práctica artística de interpretación y de composición.

En la investigación artística hay una marcada idea de que un bucle de retroalimentación entre la creación y la reflexión interviene en sus procesos. Por ejemplo, el bucle acción/creación-reflexión definido por López-Cano y San Cristóbal (2014), como un "hacer y reflexionar constantemente" (p. 77); o el tándem de aspectos teóricos, técnicos y creativos dentro de la investigación-creación descrito por Chapman y Sawchuck (2012).

Estos procesos guardan similitud con los ciclos de retroalimentación entre pensamiento, codificación y escucha del live coding propuestos por McLean (2011) y Baalman (2015), mencionados a mitad de esta sección. Esta comparación me ha llevado a reflexionar, junto a Aarón Escobar, acerca de una estrategia en la que el conocimiento circula en "un ciclo de programación, creación e investigación" (Escobar y Villaseñor en García y Silva Treviño, 2022, p. 67), lo cual nos permite lidiar con diferentes ejes involucrados en la investigación artística en tecnología musical.

A partir de lo anterior, mi postura es que realizar live coding debe involucrar a la práctica artística y al desarrollo tecnológico, pues de esta manera es posible accesar a la promesa de lo abierto en el código y el conocimiento para empujar una forma de expresión desde la base de la estructura computacional. Esta incursión se da en un proceso de transposición que comienza en la exploración de la práctica artística y se encamina hacia la prescripción del desarrollo tecnológico, aunque a veces ocurre al revés ocasionando así una retroalimentación en ambos sentidos. Cabe mencionar que la ruta que sigue mi investigación es la que va de la música por computadora a las ciencias de la computación. Con esto no propongo una ruta que lleva a quien

practica live coding desde su práctica artística al desarrollo tecnológico, sino una en la que circula en ambos sentidos con el propósito de problematizar su práctica artística. Esta ruta tiene la particularidad de transitar entre diferentes estratos artísticos y tecnológicos como veremos en el siguiente capítulo.

# 3 Apertura y abstracción en la práctica del live coding

La apertura y la abstracción son dos características de los lenguajes de programación que, por un lado, permiten realizar live coding con un lenguaje de alto nivel y, por otro, ingresar a un nivel más bajo del lenguaje para analizar sus componentes. Esto se traduce en la posibilidad de acceso a diferentes campos de conocimiento del live coding guiados por características técnicas y culturales del los lenguajes de programación. Con ello, se puede reflexionar sobre los procesos de programación en la creación artística y el desarrollo de software para transponerlos en la práctica del live coding.

El análisis de la transposición entre las formas de programación propuesto en esta sección ocurre en el plano del entendimiento humano, antes del proceso de traducción del código fuente en lenguaje de máquina. Lo anterior apunta a la noción de abstracción, la cual nos dice de manera general que no es necesario entender los procesos de cómputo que ocurren debajo mientras programamos. Esta característica nos permite manejar la complejidad de los procesos de cómputo como bloques o cajas cerradas que encapsulan las múltiples piezas involucradas en el momento de programar. En algunos casos, el código fuente de esas cajas es declarado abierto por quienes programaron el lenguaje de programación o alguna pieza de software. Entonces, si lo requerimos, podemos accesar a su contenido para observar los componentes y desglosar cada pieza para estudiarla en detalle. Esto nos lleva a la metáfora del descenso que se da con la abstracción, esto es, bajar y abrir las pequeñas cajas que nos guían por los detalles del software y, en algunos casos, nos acercan a los componentes físicos de la computadora.

Para ingresar a los componentes desglosados de un lenguaje de programación no solo se requiere la apertura de su código fuente, también el conocimiento que nos permite entender sus estructuras. Esto sucede en un contexto en que la práctica de arte computacional está imbricada con su desarrollo tecnológico por lo que, para poder analizar esta relación se hace necesario reconocer y separar las partes que participan.

Abrir el código y el conocimiento de un lenguaje de programación permite su uso y encamina a cada practicante a que eventualmente desarrolle alguna parte del mismo. Los conocimientos de programación para crear música y para desarrollar un lenguaje de programación, o parte de él, comparten saberes pero se organizan de manera distinta. Por un lado, la programación en el live coding musical nos enfrenta a conocimientos informáticos y musicales que son aplicados por dinámicas de prueba y error en el momento de programar. Por otro lado, la programación de un objeto computacional, sea este un lenguaje de programación, una clase, módulo o función, demanda una planeación en la que la prueba y el error se vuelven parte de los pasos a seguir de una metodología o flujo de trabajo.

Con base en lo anterior, este capítulo vincula a la práctica artística y al desarrollo tecnológico con las características de abstracción y apertura de los lenguajes de programación. La función que cumple es reconocer las partes de la relación observada para, a partir del análisis, proponer una forma de transposición entre las formas de programar en el live coding que permita dar cuenta de las transformaciones que experimenta la práctica artística en el caso particular de SonoTexto.

## 3.1 El lenguaje de programación y sus niveles de abstracción

Programar requiere el conocimiento de un *lenguaje de programación*, el cual opera arriba de distintos componentes de cómputo como pueden ser otros lenguajes de programación, compiladores, intérpretes y componentes hardware. Esta situación no requiere el entendimiento de las operaciones internas que realiza cada componente para programar con un lenguaje de programación determinado. En la computación, dicha particularidad se relaciona con la noción de *abstracción*. Para entender la noción de abstracción en la computación primero se necesita exponer que se entiende por un lenguaje de programación ya que es ahí donde puede ejemplificarse.

Un lenguaje de programación, según Daniel Temkin (2017), es un término difícil de precisar y su definición generalmente es formulada desde su funcionalidad. El autor lo describe como un lenguaje con el que podemos expresar comandos o instrucciones a una computadora. Temkin subraya que los lenguajes de programación son lenguajes formales cuya sintaxis y semántica son precisas ya que no presentan ambigüedad para que tanto el programador como la computadora entiendan lo que hace cada programa. Para Glenn Brookshear (2012), un lenguaje de programación facilita al humano la lectura y la escritura de código fuente al mismo tiempo que permite a la computadora traducirlo a su propio lenguaje. Por su parte, Joyanes Aguilar (2020) coincide en que un lenguaje de programación es un tipo de notación especial con sintaxis y semántica precisas. Con este, añade el autor, es posible escribir programas de cómputo que consisten en instrucciones secuenciales derivadas de la codificación de algoritmos que resuelven los problemas computacionales que plantea cada programa.

Respecto a la abstracción, Joyanes Aguilar (2020) comenta que esta nos permite manejar la complejidad de los objetos en el mundo real de tal forma que los vemos como entidades definidas con comportamientos propios y no como el conjunto de componentes que los forman. Asimismo, añade que la abstracción se descompone en diferentes grados de complejidad llamados *niveles de abstracción*. En la computación, continúa el autor, la abstracción permite construir, analizar y gestionar sistemas complejos sin ir a sus detalles. Sumando a lo anterior, Abelson et al. (1996) se refieren a los procesos computacionales que ocurren en las computadoras como entes abstractos. Los autores dicen que el mecanismo de los lenguajes de programación llamado abstracción consiste en elementos compuestos que son manipulados como unidades.

Con lo anterior, estos autores nos dan a entender que la abstracción nos permite pensar un programa de cómputo como una unidad compuesta por bloques con distintas funcionalidades que, a su vez, están constituidos por pequeños mecanismos. Tal estructura, vista de lo general a lo particular, crece en complejidad conforme se desarma el programa en unidades más pequeñas. Bajo este panorama, los bloques de código que nos muestran el funcionamiento general del programa se encuentran en un nivel de abstracción alto. En sentido inverso, las pequeñas piezas de código que conforman dichos bloques se encuentran en un nivel de abstracción más bajo. En

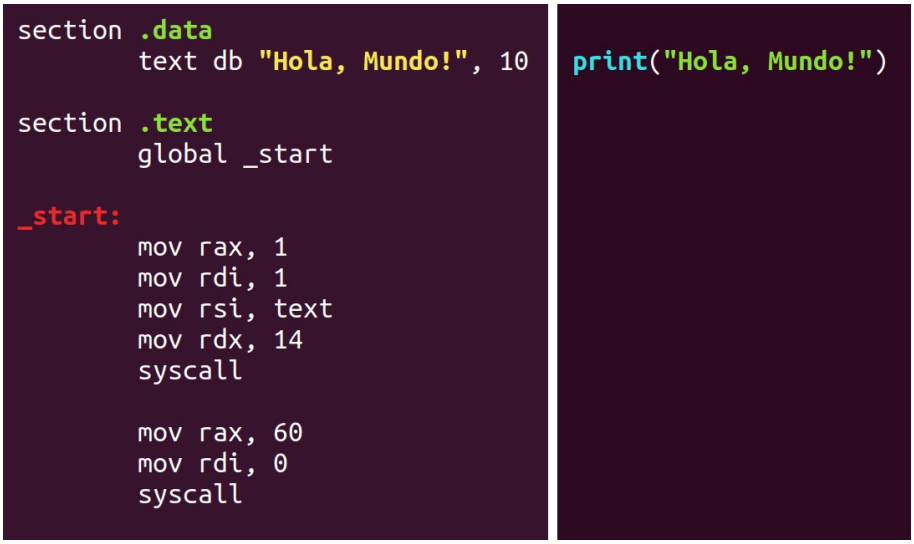
este sentido, la noción de abstracción es aplicada a los lenguaje de programación para asignarles la cualidad de alto nivel cuando su sintaxis es cercana al entendimiento humano y de bajo nivel cuando se acerca más al de la máquina.

La razón de esta división obedece a la necesidad de facilitar al humano la escritura de secuencias numéricas requeridas para programar una computadora, lo que implica diseñar los lenguajes de programación en niveles de abstracción altos. Según Brookshear (2012), el camino para hacer más fácil la escritura y lectura con los lenguajes de programación se remonta a los años 1940 cuando su diseño pasó de la programación numérica en lenguaje de máquina a la programación basada en mnemónicos para finalmente llegar a la inclusión de palabras o *tokens*[49]. Dentro de este proceso de transformación tecnológica de los lenguajes de programación, Marino (2020) discute el caso de Flow-Matic. Dicho lenguaje de programación fue desarrollado por Grace Hopper en 1958, junto a un equipo de programadores, con la particularidad de incluir palabras del inglés que lo acercaran a una sintaxis del lenguaje natural contrario a la notación científica y matemática que imperaba en ese momento. El objetivo de implementar una sintaxis que incluía palabras del inglés a un lenguaje de programación era enseñar a programar con facilidad a empresarios y militares no programadores. Marino comenta que este caso es el inicio de la tendencia a desarrollar lenguajes de programación de alto nivel, no sin antes discutir las implicaciones que conlleva acercar la programación al lenguaje humano, la cual, por un lado, facilita la programación a quienes entienden la lengua inglesa pero, por otro, puede excluir en cierto grado a quienes no.

Los niveles de programación alto y bajo en el contexto de los lenguajes de programación han sido definidos de diferentes maneras. Por ejemplo, Bolanakis et al. (2011) mencionan que los lenguajes de bajo nivel, refiriéndose a los lenguajes ensamblador, suelen ser difíciles de aprender por su carencia de estructuras. Ante ello, una ventaja que mencionan es que la programación de bajo nivel con lenguajes ensamblador permite conocer de cerca los elementos del hardware de

49 Aquí la palabra *token* se refiere a la asignación de una palabra a una función de la programación, tiene significado para el humano pero no para la computadora. Según Marino (2020), los *tokens* no son procesados como lenguaje natural por la computadora, estos simplemente relacionan una función con una palabra que le permite al humano entender el tipo de instrucción que va a programar. Por ejemplo, el nombre asignado a la función *print* de Python podría ser cambiado por imprimir y la computadora realizará la misma acción, mientras que el programador requiere conocer el significado de la palabra en español.

una computadora. La enseñanza de los lenguajes de bajo nivel, según los autores, viene de los años 1980 aunque dicen que en la actualidad los currículos escolares y el avance de la computación han privilegiado la programación de alto nivel convirtiendo al hardware en un dispositivo desconocido y oculto. Por otro lado, los autores mencionan que los lenguajes de alto nivel facilitan la programación del hardware al ser escritos con símbolos reconocibles por el humano como palabras clave, letras del alfabeto y marcas de puntuación. Esto implica que un compilador, o programa traductor de código, deberá traducir el código escrito por el programador al lenguaje de la computadora. Por esta razón no es necesario conocer el funcionamiento del hardware ya que el paso intermedio de traducción libera al programador de lidiar directamente con el procesador de la computadora.



```
section .data
        text db "Hola, Mundo!", 10

section .text
        global _start

_start:
        mov rax, 1
        mov rdi, 1
        mov rsi, text
        mov rdx, 14
        syscall

        mov rax, 60
        mov rdi, 0
        syscall
```

```
print("Hola, Mundo!")
```

*Figura 3.1. A la izquierda: instrucción que imprime la frase "Hola, Mundo!" escrita con el ensamblador Nasm de bajo nivel, transcrita del video 'x86_64 Linux Assembly #1 – "Hello World!"' del canal de YouTube kupala. A la derecha: misma instrucción con el lenguaje Python de alto nivel.*

Lo anterior plantea dos aproximaciones de la abstracción. La primera es una programación cercana a los elementos físicos de la computadora con un lenguaje de bajo nivel como es el caso de los lenguajes ensambladores, lo que David Chisnall (2018) llama "estar cerca

del metal” (p. 2). Esto implica, por un lado, utilizar lenguajes de programación fáciles de traducir por la computadora y, por otro, facilitar la escritura de código fuente para que el humano programe con lenguajes de programación “parecidos” al lenguaje natural. Lo que interesa a este estudio es la posibilidad de observar desglosados algunos de los componentes de un lenguaje de alto nivel para reconocer las piezas que nos permitirán desarrollar nuestro objeto computacional y no tanto llegar hasta el hardware.

Respecto a lo anterior, Joyanes Aguilar (2020) menciona los dos puntos de abstracción anteriores. De manera general, el autor dice que el lenguaje humano es considerado de alto nivel mientras que el lenguaje máquina de bajo nivel. Entre estos puntos, el autor describe el lenguaje de máquina, o código máquina, como un lenguaje binario conformado por 0 y 1, definido por la arquitectura de cada computadora y dependiente de esta. Cuenta que originalmente las computadoras eran programadas con lenguaje binario. Después, menciona que llegó el lenguaje ensamblador que define como un lenguaje que utiliza palabras abreviadas del inglés llamadas mnemónicos que representan operaciones básicas[50]. A diferencia del lenguaje binario, el código fuente escrito con lenguaje ensamblador requiere ser traducido por un traductor llamado ensamblador. Finalmente, dice que los lenguajes de programación de alto nivel permiten escribir sentencias facilitando la programación respecto a los lenguajes anteriores por su cercanía a los lenguajes naturales y a la notación matemática. Al igual que los lenguajes ensambladores, los lenguajes de alto nivel requieren ser traducidos por *programas compiladores* o los programas *intérpretes*.

Como se menciona en el párrafo anterior, hay dos técnicas para traducir el código fuente escrito por un humano a uno que entiende la computadora para ejecutarlo: la compilación y la interpretación. La traducción del código fuente con un *programa compilador* nos proporciona un archivo ejecutable que la computadora emplea para correr un programa entero. Según Brookshear (2012) el proceso de compilación crea una copia en lenguaje máquina que se utiliza de manera posterior. Mientras que, la técnica empleada por los *programas intérpretes* consiste en traducir el código fuente por partes, es decir, el programa ejecuta las instrucciones conforme se van traduciendo (Brookshear, 2012). De estas tecnologías, un programa intérprete resulta mejor

50 En la *Figura 3.1* se observan los menmónicos que son palabras cortas como mov, rax, rdi, etc.

para el live coding ya que posibilita la traducción del código fuente por secciones sin la necesidad de parar el funcionamiento del programa para actualizarlo.

Bajo lo antes expuesto, el proyecto de live coding SonoTexto emplea un lenguaje de programación de alto nivel e interpretado. La abstracción sirve como una herramienta de análisis, como se verá más adelante, que permite desglosar los componentes encapsulados de un lenguaje de programación de alto nivel. Si bien esta decisión me aleja de estudiar los procesos computacionales cercanos al hardware, por otro lado me acerca al pensamiento musical que puede ser expresado con los lenguajes de programación de alto nivel. En este sentido, el live coding puede ser visto como una actividad de programación realizada en la parte superior de una serie de estratos computacionales de los que solo requerimos entender el funcionamiento del estrato más alto. No obstante, como artistas podemos indagar en el funcionamiento de los estratos inferiores por diferentes motivos técnicos y estéticos; ya sea para diseñar y desarrollar un objeto computacional o para explorar los lenguajes de bajo nivel a nivel creativo[51].

La abstracción presente en los lenguajes de programación coloca la escritura exploratoria del live coding en un terreno de expresión humana, asimismo muestra la relación que hay entre el código fuente que escribimos y las acciones que lleva a cabo la computadora al traducirlo y ejecutarlo. Para aprovechar esta posibilidad, algunas comunidades y movimientos, como el software libre, el código abierto y el propio live coding, abogan por la apertura y muestra del código fuente con el objetivo de desmitificar y transparentar la programación. Como veremos en el siguiente capítulo con SonoTexto esto no sugiere necesariamente llegar a los niveles más bajos del cómputo, como en el caso del lenguaje ensamblador o incluso el código máquina, sino de abrir algunas partes del código fuente encapsuladas en el nivel de abstracción alto que lo prescribe. Esto con el objetivo de entender el funcionamiento de los niveles y capas cercanas a la práctica artística del live coding que posibilitarían un descenso paulatino de niveles de abstracción. Tal acto de apertura, sin embargo, no implica el acceso inmediato a la programación prescriptiva, pues ello depende de un conocimiento especializado que nos ayude a comprender las estructuras del código fuente en los diferentes niveles de abstracción a los que lleguemos. En

51 Por ejemplo la pieza *BetaBlocker* (2006) de Till Bovermann y Dave Griffiths (2004) https://youtu.be/19iAW_YuhcY o las participaciones audiovisuales de Julio Zaldívar en las Sesiones de Live Coding del Cmm en las que programaba el microcontrolador ATTINY2313 con lenguaje C; véase un ejemplo sonoro https://vimeo.com/55453484 y un ejemplo visual https://vimeo.com/48488075.

este sentido, uno de los objetivos que tiene el ingresar al código fuente abierto es construir una vía de acceso al conocimiento del software y a la programación para la creación artística vinculada a su desarrollo tecnológico.

## 3.2 Apertura del código fuente

Abrir el código fuente y mostrarlo proyectado son consignas relacionadas al software libre, el código abierto, la cultura hacker y el live coding. Esta última práctica ha sido influenciada por las tres primeras corrientes en su propósito de proporcionar el conocimiento para que cualquiera pueda crear arte con una computadora en el contexto de una presentación en vivo.

Los lenguajes de programación para el live coding son diseñados para operar en un nivel alto de abstracción que permite escribir y entender fácilmente el código fuente en el momento de realizar live coding con ellos. Esa facilidad es un tipo de apertura que permite el ingreso a la programación en el contexto del live coding cuando no se cuenta con conocimientos previos de programación o incluso de música, imagen o matemáticas. Lo anterior se observa, por ejemplo, en el inicio a la programación exploratoria con los programas para live coding musical y visual Sonic Pi e Hydra, que desde las primeras líneas de código escritas con ellos muestran su sencillez. Esto lo confirma Jack (en Piranha Lab, 2019), quien menciona que una de sus preocupaciones al desarrollar Hydra fue facilitar no solo su uso sino el hackeo del mismo, puesto que el lenguaje de programación con el que está desarrollado (JavaScript) es el mismo con que se programa. Esto abre la posibilidad de involucrarse con el lenguaje de programación desde el primer momento y, añade, asumirse como programador.

En este sentido, facilitar la escritura de código con un lenguaje de programación es una forma de apertura que da acceso a través del diseño simple. En el caso de SonoTexto, más adelante veremos que la cantidad de las líneas de código requeridas para realizar las funciones de grabación, reproducción y secuencia de sonido son mínimas. Un diseño que parte de lo simple y

que resuena con la idea de poner al alcance una serie de funciones que permiten programar de manera inmediata a quien inicia a realizar live coding con sonido.

El acceso a la tecnología, los repositorios y la documentación permiten construir nuevos objetos computacionales de expresión musical. Esto, sin embargo, sucede desde la óptica del desarrollo tecnológico. Pero, en un contexto de creación artística basada en la programación exploratoria ¿por qué tendríamos que accesar al nivel de desarrollo del software? ¿por qué no simplemente utilizarlo para la creación?

La respuesta es compleja, primero porque la invitación de lo abierto está hecha, segundo porque a menudo los lenguajes de programación dedicados al arte parten de problemáticas y aproximaciones estéticas de sus desarrolladores, mismas que se extienden a través del software a los usuarios y, tercero porque al ver cómo está estructurado un software podemos generar ideas para realizar nuestros propios desarrollos bajo la idea de construirlos sobre tecnologías abiertas previamente desarrolladas. Estas razones apuntan a desarrollos con intenciones tecnológicas que reflejan posturas políticas y estéticas.

En este sentido, no hay que confundir el código mostrado en las presentaciones de live coding con la visibilidad que nos permite el acceso al código fuente de un software almacenado en un repositorio. El primero es resultado de la programación exploratoria y el segundo de la programación prescriptiva. El código proyectado que vemos en el concierto de live coding es una interfaz que posibilita crear música e imagen mientras que el código fuente del software contenido en repositorios es el mecanismo de funcionamiento.

La apertura permite ver el código fuente de un programa desde un repositorio, observar la materia de la obra artística computacional y compartir información. Lo anterior pasa a través de la lente de nuestro conocimiento. De esta manera, parte de los objetivos de abrir y mostrar el código fuente en el live coding son los de observar los mecanismos del software, la materia de la obra digital y los de aprender a programar para expresarnos. Tal apertura viene acompañada con los niveles de abstracción, lo que permite, por un lado, programar una computadora con facilidad y poco conocimiento en el momento de expresarnos y, por otro lado, seguir reglas que prescriben cómo se programa un software. El acceso a la programación exploratoria de una computadora es

importante para introducirnos a ciertas tecnologías pero no es suficiente pues, si bien nos deja en la posición de usar un software, no nos permite participar en su diseño de manera inmediata. En este sentido, la apertura del código fuente nos da acceso tanto a la expresión creativa de la programación exploratoria como a la posibilidad de bajar niveles de abstracción para entender los elementos que componen el entramado informático en el que se sustenta el arte computacional que realizamos. El descenso de niveles nos permite entender mejor el desarrollo de software en un sentido prescriptivo. Dicho paso puede ser simple pero no deja de ser significativo para entender a profundidad la tecnología que usamos. En este contexto, la apertura nos encamina a desarrollar una postura crítica que parte de la reflexión sobre los conocimientos de programación a nivel de expresión y desarrollo.

## 3.3 Alto, bajo, abierto

La práctica artística del live coding se realiza con lenguajes de programación que facilitan la escritura, lectura y traducción del código fuente durante una presentación. Durante el proceso de programación basta conocer algunos objetos, funciones y formas de secuenciar para comenzar a crear música. A su vez, estas características facilitan la iniciación a la creación musical con lenguajes de programación.

En específico, algunos desarrollos tecnológicos del diseño de los lenguajes de programación que facilitan el live coding son la abstracción de alto nivel, la técnica de traducción interpretada, así como los paradigmas de programación orientado a objetos, funcional y declarativo. Tales desarrollos, además de resolver la comunicación entre el programador y la computadora permiten llevar a cabo las tareas artísticas. En este sentido, cuando creamos música con un lenguaje de programación operamos arriba de un entramado de planteamientos, soluciones y decisiones tomadas por quienes desarrollan una tecnología que nos permite programar nuestras ideas musicales. Una vez que estamos involucrados en el live coding, el proceso de creación artística nos confronta a la necesidad de diseñar y desarrollar nuestras piezas de software. Esto implica entender las estructuras sobre las que operamos. Entonces, bajo un

proceso de ingeniería inversa comenzamos a descender niveles de abstracción para abrir las unidades del lenguaje de programación que utilizamos para ver cómo están hechas y, con base en ellas, programar nuestros componentes.

Mi intención al exponer los niveles de abstracción en los lenguaje de programación y la apertura de código fuente en el live coding ha sido explicar los elementos que conforman el espacio de análisis y acción que delimita mi investigación. Para visualizar dicho espacio he diseñado la gráfica *alto, bajo, abierto* en la que coloco los términos expuestos con relación a la *práctica artística*, el *desarrollo tecnológico* y el *código fuente abierto*. La gráfica plantea que los niveles de abstracción alto y bajo están vinculados con la práctica artística y el desarrollo tecnológico dentro de un contexto de apertura del código fuente. Para mostrar dicho vínculo he asignado la categoría de *capas* a la práctica artística y al desarrollo tecnológico, mientras que los niveles de abstracción mantienen su categoría. A continuación se muestra y describe la gráfica.

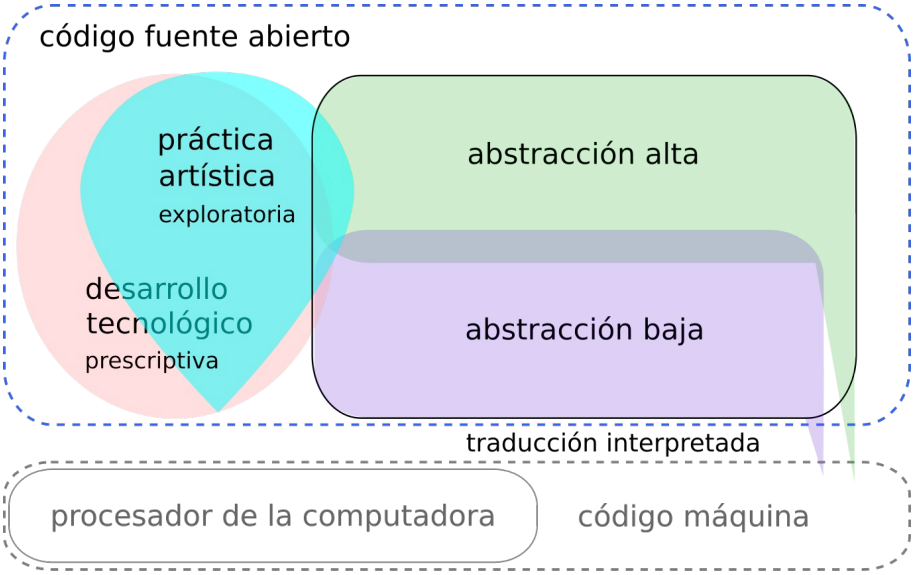



*Figura 3.2. Alto, bajo, abierto.*

El *código fuente abierto* se encuentra en la parte superior izquierda de la gráfica. Este se refiere al código que escribe el humano. Está escrito por quien realiza live coding o por quien

desarrolla el software para ello. Opera en un contexto de circulación de conocimiento sobre programación exploratoria y prescriptiva. La cualidad *abierto* posibilita dos cosas en momentos distintos: mostrar el código exploratorio a la audiencia durante una presentación de live coding y desencapsular el código fuente prescriptivo para ver cómo está programado.

La capa de la *práctica artística* se encuentra al frente en la parte izquierda de la gráfica. Esta capa engloba la programación exploratoria u holística del live coding y opera, en mayor medida, en un nivel alto de abstracción y en menor medida en un nivel bajo.

La capa del *desarrollo tecnológico* se encuentra en la parte izquierda de la gráfica al fondo. Engloba la programación prescriptiva de los objetos computacionales para hacer live coding como pueden ser un lenguaje de programación, una clase, un *plug-in* o una unidad generadora de sonido. Opera en ambos niveles de abstracción.

El nivel de *abstracción alto* se encuentra en la parte derecha de la gráfica hacia arriba. Se refiere a la legibilidad y facilidad de codificación cercano al entendimiento del lenguaje humano y al lenguaje de programación encapsulado que facilita la escritura de código fuente durante una presentación de live coding.

El nivel de *abstracción bajo* se encuentra en la parte derecha de la gráfica debajo del nivel de abstracción alto y antes de la traducción interpretada del código fuente. Se refiere al código fuente con el que están hechos los lenguajes de programación y tiende a estar más cerca del lenguaje de la máquina.

La *traducción interpretada* se refiere a la técnica de traducción del código fuente a código máquina realizada por un programa intérprete que permite traducir el código fuente de un programa instrucción por instrucción. Debido a su flexibilidad e interacción al momento de programar, esta técnica de traducción es la que utilizan los lenguajes de programación de live coding para mantener una salida de sonido o imagen continua cuando se realizan cambios en el código fuente de la programación exploratoria.

El *código máquina* está colocado en la parte inferior derecha de la gráfica. Es el código traducido a lenguaje binario que entiende el hardware de la computadora y que utilizará para ejecutar las tareas descritas en el código fuente escrito por el humano. Este código, después de

ser traducido, pasa por un proceso interno para convertirse en el código que finalmente ejecuta la computadora.

El *procesador de la computadora* está colocado en la parte inferior izquierda de la gráfica. Representa el componente CPU de la computadora que se encarga de gestionar el código máquina de y hacia otros componentes para que la computadora realice las acciones descritas.

Lo que está enmarcado por la línea azul punteada de la gráfica señala el contexto humano de la programación que sucede antes del punto de la traducción a código máquina. Por su parte, el área gris enmarca el ámbito de la máquina que recibe el código interpretado para ejecutarlo. Cabe aclarar que la investigación no se enfoca en la sección gris, sin embargo, está señalada para indicar el camino que sigue la escritura de código fuente dentro de la computadora.

La intención de colocar los niveles de abstracción junto a las capas de práctica artística y desarrollo tecnológico, es visualizar la posibilidad de descender a los componentes encapsulados del lenguaje de programación y no tanto señalar la cercanía con los componentes hardware de la computadora. De esta manera, es posible indicar que la transposición de saberes entre la práctica artística y el desarrollo tecnológico está situada en los niveles de abstracción.

En la gráfica, la práctica artística está vinculada principalmente al nivel de abstracción alto, esto señala que los componentes con los que se realiza la programación exploratoria en el live coding están encapsulados. Accedemos a ellos durante la programación exploratoria del live coding por medio de palabras, mensajes o comandos que llaman a sus funciones internas. Por otro lado, el desarrollo tecnológico está vinculado a ambos niveles. Esto muestra que la programación prescriptiva de objetos computacionales ocurre en diferentes niveles de abstracción que enlazan los diferentes estratos tecnológicos de la creación artística.

Lo que planteo con ello es que la apertura del código mostrado en las presentaciones de live coding y el código fuente abierto de los lenguajes de programación colocado en sus repositorios está vinculada a la abstracción computacional. Por un lado, el nivel alto de programación empleado por quien programa con el software y, por otro lado, el nivel bajo de programación por quien lo desarrolla. En este sentido, una abstracción encapsula componentes

que delegan la complejidad al código empacado y, de esta manera, permite la expresión artística en el alto nivel.

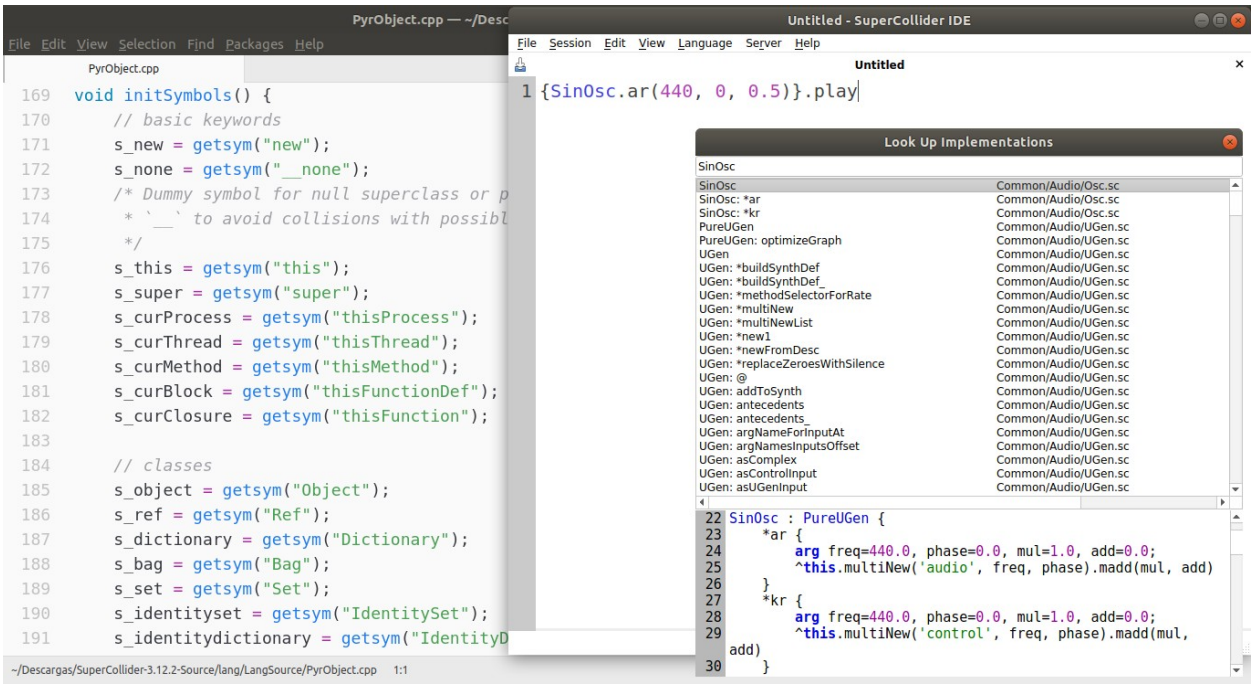



*Figura 3.3.* Código fuente de SuperCollider escrito con C++ (izquierda), nivel de desarrollo con sclang (derecha abajo) y nivel de práctica artística con SuperCollider (derecha arriba).

La *Figura 3.3* muestra diferentes niveles de abstracción que encapsulan partes del lenguaje de programación de SuperCollider. Primero, el nivel de abstracción más alto está situado en la parte alta derecha que muestra una onda sinusoidal de 440 hertz. Ese es el nivel alto del lenguaje de programación interpretado sclang con el que el live coder programa de manera exploratoria.

En segundo lugar, colocado en la parte derecha baja de la imagen, se encuentra el nivel de abstracción que encapsula el código fuente de la clase `SinOsc`. El código fuente mostrado en la ventana de implementaciones está escrito con el mismo lenguaje sclang, aquí se muestra la forma prescriptiva del código. Al abrir esta ventana que muestra el código de una clase podemos

analizar sus componentes internos aunque, como he mencionado antes, no necesitamos conocer su contenido para crear música en el nivel más alto mostrado en el primer nivel de la figura.

Finalmente, del lado izquierdo de la imagen se encuentra un nivel de abstracción más bajo que encapsula código fuente del desarrollo tecnológico del lenguaje de programación de SuperCollider. La imagen muestra parte del código fuente del archivo PyrObject.cpp escrito con el lenguaje de programación C++. Este es uno de los tantos archivos que forman parte de la carpeta *LangSource*, la que a su vez es parte de la carpeta *lang*. Esta última carpeta junto a otras tantas (véase *Figura 4.3*) conforman el código fuente de SuperCollider. Desde la óptica del live coding, esta parte del programa está "cerrada". Al compilar el programa, estos archivos de código fuente construyen el lenguaje sclang con el que podemos programar de forma exploratoria, el servidor que realiza la parte de síntesis y el entorno gráfico de programación donde vemos el código fuente que escribimos.

De esta manera, el rango de abstracción de SuperCollider se despliega desde la escritura de código para la expresión artística en su parte más alta hasta el código que conforma el diseño y desarrollo del lenguaje de programación. El ejemplo de la *Figura 3.3* muestra dos lenguajes de programación (sclang y C++) involucrados en el conglomerado de piezas de software que componen a SuperCollider. La intención de mostrar así esas secciones del código fuente del programa no es marcar una jerarquía entre capas y niveles, sino señalar dónde ocurren las formas de programación exploratoria y prescriptiva, así como los vínculos dentro del entramado de la práctica artística y el desarrollo tecnológico. Así, el diseño de SuperCollider escrito con el lenguaje de programación C++ está situado en la capa de desarrollo tecnológico y en el nivel de abstracción más bajo del programa mientras que, el código que se escribe durante la presentación de live coding con el lenguaje de programación sclang se encuentra en la capa de práctica artística y en el nivel de abstracción más alto del programa.

De esta manera, la programación exploratoria de la práctica artística se escribe sobre la capa de desarrollo tecnológico que previamente ha sido programada. El código fuente escrito durante la presentación de live coding es la primera capa que vemos y con la que interactuamos en el momento de programar música. Está cargada de conocimientos musicales, computacionales y culturales. Mientras que la programación prescriptiva se encuentra operando en la capa de

abajo con las funciones encapsuladas que conforman el programa. Es la encargada de implementar, en el diseño del lenguaje de programación, los elementos que permiten formular las estructuras musicales.

Luego, hay un punto en el que convergen ambas capas y niveles que está precisamente en el lenguaje de programación sclang. Por un lado, las clases del programa escritas con sclang colocan esta sección en la capa de desarrollo con un grado de abstracción bajo en el sentido que encapsulan sus funciones. Por otro lado, como se mencionó arriba, el código fuente escrito durante las presentaciones de live coding opera en la capa de la práctica artística con un grado de abstracción alta. Este punto es donde se centra mi análisis pues permite ver claramente cómo un mismo lenguaje de programación es utilizado de manera exploratoria y prescriptiva y cómo estos modos de uso o formas de programar se relacionan entre sí.

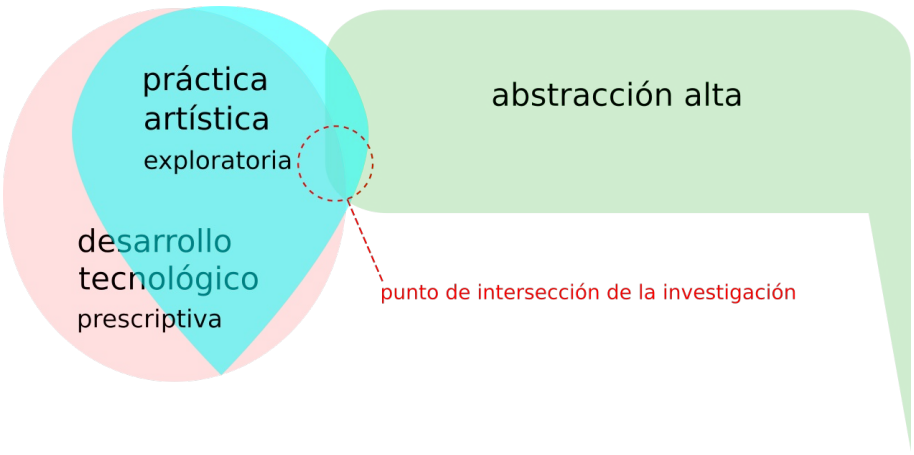



*Figura 3.4. Punto de intersección de la investigación.*

Con base en lo anterior, la problemática de la investigación puede visualizarse en la parte izquierda de la gráfica alto, bajo, abierto como lo indica el extracto presentado en la *Figura 3.4*. La imagen indica el lugar preciso de la intersección de las capas de la programación exploratoria y prescriptiva con el nivel de abstracción alta. Ese punto es el acceso abierto a la capa de desarrollo tecnológico que permite el diseño de formas de expresión que han desembocado en el diseño de SonoTexto, como expondré a detalle en las siguientes páginas. A su vez, es el punto

donde se puede observar cómo ocurre la transposición siempre que el código fuente de ambas capas se mantenga abierto. En el caso de esta investigación, la transposición entre capas analizada ocurre en el elemento *clase* del lenguaje de programación SuperCollider como veremos en el siguiente capítulo.

La *Figura 3.4*, muestra que el espacio de análisis que vincula a la práctica artística con el desarrollo tecnológico contiene a las formas de programación exploratoria y prescriptiva en un área donde se superponen ambas capas con la abstracción alta del programa. Esta intersección es importante señalarla pues conecta la expresión artística con los mecanismos tecnológicos desarrollados e implementados en los objetos computacionales para hacer live coding. De esta manera, si podemos diseñar mecanismos de expresión dentro del lenguaje de programación empleado, además de saber cómo expresarnos con código fuente, damos un paso que va de usar un software para programar música a uno para diseñar parte de sus componentes, algo que nos permite entender mejor una tecnología y entrar a las discusiones en torno a ella desde otra perspectiva.

# 4 Desarrollo tecnológico, investigación y práctica artística en el proyecto SonoTexto

El inicio del live coding en México estuvo influenciado por la modalidad de programación *from scratch*. En esa situación, aunque el live coder parte de una pantalla en blanco, las líneas que escribe son estructuras que ha ensayado, memorizado, in-corporado o explorado de manera previa. Por otro lado, escribir código desde cero implica trabajar sobre una serie de desarrollos tecnológicos previamente programados como el caso de un lenguaje de programación y sus extensiones. Por tal razón, la modalidad *from scratch* no la planteo aquí como un acto virtuoso en el que la creación musical implica partir de la nada o como valor de "autenticidad" del live coding frente a las presentaciones que parten con código preparado. Dicha modalidad la entiendo desde la práctica colectiva de live coding fomentada en México por el Taller de Audio del Cmm (véase § Introducción) que se basaba en la postura de no respaldar el código fuente de las presentaciones expresada en el Manifesto de Toplap. En todo caso, el live coding *from scratch* parte del código fuente implementado en los lenguajes de programación y su extensiones.

El enfoque anterior de cierta manera se ve reflejado en el trabajo de quienes participamos en las Sesiones de Live Coding organizadas por el Cmm en la primera mitad de los años 2010. En mi investigación esto se observa en el estudio de caso del proyecto de live coding SonoTexto, el cual es desarrollado en varios de sus aspectos durante el capítulo. Este proyecto está formado por un objeto computacional diseñado para realizar live coding con sonido grabado en el momento de una presentación y por la serie de presentaciones realizadas con dicho objeto. Tanto el objeto computacional como las presentaciones reflejan la aproximación de iniciar desde cero el código fuente escrito y el sonido utilizado como material de creación. Podría pensarse que la propuesta de iniciar el sonido desde cero en una presentación de live coding es redundante, pues se espera que el código fuente sea el que lo genere o active con las instrucciones que se escriben en tal evento. En el caso de SonoTexto, tal planteamiento parte de la idea de trabajar con el sonido del espacio acústico donde se lleva a cabo la presentación de live coding, un sonido que no existe aún pero que irá sucediendo mientras se desenvuelve el evento. Para ello, el objeto

computacional graba sonido que sucede en el momento dentro de la memoria volátil de la computadora. De esta manera, la idea de comenzar de cero con la programación se traslada también al sonido que, junto al código fuente escrito, desaparece al apagar la computadora.

El objeto computacional SonoTexto, así como las presentaciones de live coding realizadas, han sido formuladas y realizadas en el contexto de la presente investigación. El primero es referido como el eje de desarrollo tecnológico y las segundas como el eje de la práctica artística. El objeto computacional está formado por una serie de clases escritas con el programa SuperCollider, mientras que la práctica artística consiste en distintas presentaciones de live coding, de manera presencial y en línea, con las clases SonoTexto operando en el entrono de SuperCollider.

Bajo este planteamiento, el capítulo expone el proyecto mencionado, su desarrollo tecnológico, la práctica artística realizada y su condición estética. Asimismo, describe los cambios que sufrió debido a la pandemia de COVID-19. Respecto a lo anterior, he tratado de mantener la narrativa temporal en las secciones del capítulo referentes a este hecho. Por un lado, algunas partes se enfocan en la perspectiva de explorar el sonido de los espacios donde se realizó cada presentación presencial, lo que ocurrió antes del confinamiento. Las presentaciones presenciales están marcadas por la idea de partir de cero y borrar el código y el sonido grabado al finalizar el evento. Por otro lado, algunas secciones del capítulo dirigen su enfoque a los cambios que provocó el confinamiento de los que el cambio más claro es el paso del concierto presencial en el espacio público a la transmisión en línea desde el espacio privado. Entre otras cosas, tal cambio provocó que la práctica de live coding con el sonido de entornos acústicos específicos grabado en el momento pasara a una práctica con sonido grabado y almacenado durante el confinamiento. Esto, aunque implicó un cambio en el diseño de las clases para adaptarse a tales circunstancias, mantuvo la problemática que relaciona a la práctica artística con su desarrollo tecnológico a través de sus formas de programar.

Lo anterior me llevó a experimentar diferentes transformaciones de la práctica artística a través del desarrollo tecnológico de las clases programadas, que a su vez experimentaron la transformación de su diseño debido a la pandemia. La propuesta de desplazar la programación exploratoria a la programación prescriptiva adquirió sentido en el ejercicio de navegar entre las

capas de implementación y aplicación artística del código fuente en diferentes circunstancias de presentación. El paso de la presentación pública a la presentación en línea provocó, además, un cambio del espacio-tiempo de la grabación, mientras que la programación desde cero se mantuvo. De esta manera, los ejes de la práctica artística y el desarrollo tecnológico fueron puestos a prueba bajo circunstancias distintas a la planeación original. Asimismo, ambos ejes mostraron cómo se adaptaron, informaron y transformaron al pasar de la idea de hacer live coding con sonido grabado en el momento (de la presentación) a la de hacerlo con sonido grabado en cualquier momento (del confinamiento).

## 4.1 SonoTexto

El proyecto de live coding SonoTexto consiste en un objeto computacional abierto y la práctica artística realizada con dicho objeto. El objeto computacional graba fragmentos de sonido en la memoria temporal de la computadora desde donde son reproducidos para realizar live coding. La idea es trabajar con diferentes tipos de sonido que ocurren en el tiempo y el espacio de una presentación en vivo. De esta manera, la práctica artística consiste en la improvisación con los sonidos grabados cuando estos son modificados a través de la escritura de código fuente con el lenguaje de programación SuperCollider.

El nombre es un juego de palabras compuesto por los términos *sonido* y *texto*. El término expresa la relación entre el sonido capturado y la textualidad del código fuente presente en la escritura de instrucciones para grabar, reproducir y manipular dicho sonido con la sintaxis del lenguaje de programación empelado. Esto se relaciona con la creación musical hecha con sistemas de notación (Young, 2015; Magnusson, 2019; Larrieu 2016); con prácticas musicales que utilizan tecnologías de grabación y reproducción sonora, como la música concreta de los años 1950, la música electroacústica y el paisaje sonoro; con prácticas artísticas y sonoras que reflexionan sobre el diálogo entre sonido y espacio (LaBelle, 2008; Zećo, 2021); y con la improvisación como práctica de creación musical.

*Figura 4.1. Tres tipos de sonidos grabados con SonoTexto: instrumentistas, sonido ambiente del recinto y paisaje sonoro. Imagen construida con capturas de pantalla de videos del archivo del autor: a la izquierda Sala Huehuecóyotl, al centro Galería El Rule y a la derecha departamento del autor. Foto del centro por Libertad Figueroa.*

La aproximación de SonoTexto al trabajo con sonido grabado es diferente al live coding realizado con sonido generado por el procesador de la computadora con técnicas de síntesis o que usa muestras de sonido previamente grabadas y postproducidas, como ocurre con Tidal Cycles y Sonic Pi[52]. El material sonoro capturado en el momento por SonoTexto se refiere a tres tipos de sonidos que pueden ocurrir durante un concierto, un ensayo o un momento de escucha: 1) el sonido ambiente del recinto o lugar donde se lleva a cabo la presentación, 2) el paisaje sonoro que colinda el espacio de la improvisación y 3) el sonido de instrumentos acústicos o electrónicos propios o de otros improvisadores en el caso de una sesión de improvisación sola o en grupo. Tales sonidos no están guardados en el disco duro sino que son grabados en el momento de cada presentación. De las tres situaciones planteadas hago énfasis, en primera instancia, al trabajo con sonido en el espacio público del concierto ya que fue el planteamiento que imperó durante la primera mitad de la investigación, antes del inicio de la pandemia de COVID-19.

52 Cabe mencionar que estos programas, además de los sonidos que incluye su versión *vanilla*, admiten ingresar archivos de sonidos propios.

Mi interés por trabajar con el sonido del espacio grabado en el momento se conecta con la idea de comenzar sin código fuente en la pantalla durante una presentación live coding *from scratch* y, en este caso, tampoco con sonido grabado previamente. Esto sucede en una situación en la que el espacio nos revela su forma y propiedades acústicas cuando se produce sonido dentro de él. Tal perspectiva se adscribe a una estética del espacio como productor sonoro y a las posibilidades que emergen al activar el "sonido del espacio" con el propio espacio. Tal exploración se observa en el trabajo de compositores y artistas sonoros como Alvin Lucier, Max Neuhaus y Maryanne Amacher a los que Brandon LaBelle (2008) se refiere en su estudio sobre la relación entre sonido y espacio.

En el caso de SonoTexto, tal situación ocurre con la captura de un fragmento de sonido del lugar donde se lleva a cabo la presentación cuando un micrófono conectado a la computadora captura sonido y un comando de programación activa el registro de sonido durante unos segundos para "llenar" un buffer en la memoria volátil de la computadora. En ese momento, el sonido recién grabado puede ser reproducido con otro comando de programación y amplificado en el sistema de sonido para detonar más sonidos dentro del espacio. En el caso de una presentación en vivo, también se incluye el sonido del público presente y los sonidos colindantes al espacio.

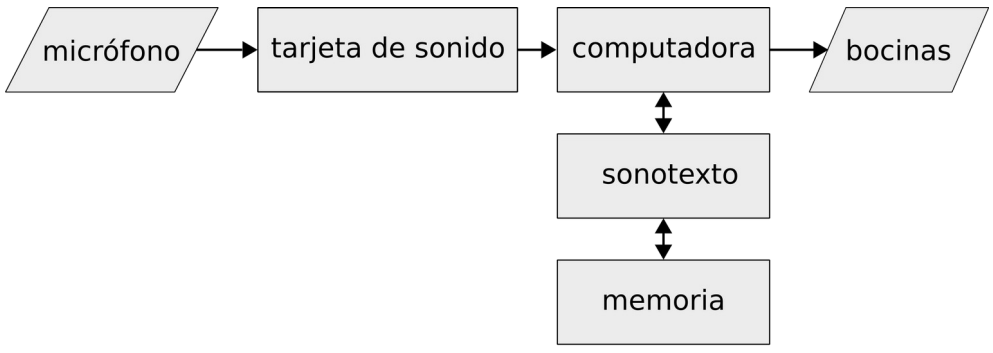



*Figura 4.2. Flujo de señal de SonoTexto.*

La grabación de sonido con esta técnica y forma de concebir la práctica artística puede extenderse a otras situaciones como mencioné anteriormente. Una es la improvisación solista o junto a otros instrumentistas donde se captura el sonido producido con sus instrumentos para incorporarlos a la improvisación del conjunto sonoro desde las posibilidades del live coding. Otra situación ocurre, como expondré más adelante, cuando la presentación de live coding en el espacio público se traslada a las actividades confinadas del espacio privado. Esto último propicia un enfoque en el trabajo con el paisaje sonoro de un entrono, con objetos sonoros y con instrumentos musicales del espacio confinado.

El planteamiento de SonoTexto, como parte de la metodología de la investigación, pone en relación la práctica artística con el desarrollo tecnológico, de tal forma que una presentación de live coding requiere de un objeto computacional para grabar y reproducir fragmentos de sonido. A su vez, el objeto computacional se diseña con base en la práctica artística y en la problemática de la investigación donde las partes se informan entre sí. Al respecto, López-Cano y San Cristóbal Opazo (2014) se refieren al proceso de investigación artística que conecta diferentes ejes como un "modo de acción creativa e indagadora que construye discurso y reflexión consciente y crítica mientras crea, abordando tanto los problemas, estéticos y técnicos, como los sociales, políticos o filosóficos" (p. 57). Mientras que Chapman y Sawchuk (2012), en el contexto de la investigación creación, mencionan que la creación, como fase integral de una investigación, implica desde el inicio reunir material, ideas, conceptos, colaboradores y tecnologías. Tal conjunto, dicen, contribuye a la investigación y producción de conocimiento y la consideran investigación en tanto actividad reveladora "habilitada a través de una percepción artística de la tecnología como práctica u oficio (o 'techné')" (p. 15).

Es decir, el objeto computacional responde a la problemática de la investigación que apunta a generar modos de expresión con la tecnología de un lenguaje de programación. Tal modo de expresión surge en mi práctica de live coding bajo la idea de "moldear el sonido de un espacio" con el código fuente. Aquí, la práctica de live coding hace referencia a otras prácticas musicales y artísticas que forman parte de mi experiencia como la música electroacústica, la improvisación libre y la instalación sonora. Esto muestra que la mezcla de conocimientos y aproximaciones musicales, estilos y estéticas no es resultado del live coding como tal, sino que

se entreteje con el bagaje de cada practicante. Lo anterior, sin dejar de observar que los lenguajes de programación para la creación musical tienen inscrita la perspectiva y cultura de quien los desarrolla.

El conjunto de ideas y prácticas que forman parte de mi práctica artística mencionadas en el párrafo anterior determinaron el desarrollo de un objeto computacional que me ha permitido no solo resolver la grabación de sonido, también incluir mi experiencia artística dentro del propio objeto computacional. Con ello, inició un proceso de retroalimentación que informó a la práctica artística y al desarrollo tecnológico en el contexto de la investigación para generar un objeto computacional con las siguientes características: 1) es parte de la práctica artística, por lo que está en proceso, 2) es idiosincrático, 3) activa la práctica artística y se informa de ella y 4) se transforma en el contexto y las circunstancias donde es desarrollado y 5) produce el conocimiento para crear otros objetos computacionales.

Este vaivén entre ideas que surgen de la práctica artística y soluciones técnicas que resuelven su modo de expresión toma forma en una pieza de código fuente estructurado que contiene las funciones necesarias para grabar con el micrófono de la computadora. Tal pieza de código está escrita bajo el paradigma de programación orientado a objetos de SuperCollider y se concreta en un objeto computacional que graba y reproduce sonido. En este sentido, la idea de representar objetos físicos dentro del ámbito digital de la computadora está detrás del paradigma de programación orientado a objetos u *object oriented programming* (OOP). Según Joyanes Aguilar (2020), los objetos, como elementos del paradigma OOP son analogías a los objetos de la vida real. La idea de este paradigma, menciona el autor, surgió para manejar problemas complejos, los cuales pueden ser descompuestos en módulos u objetos que facilitan la programación. Cada objeto está formado por datos y funciones e interactúa con otros objetos. En el contexto del paradigma OOP, el objeto encapsula datos y funciones, hereda comportamientos de otros objetos y presenta polimorfismo o distintas formas de comportamiento usando el mismo código. El encapsulamiento de código fuente tiene el objetivo de reutilizarlo en distintas circunstancias y proyectos.

En este punto es necesario desambiguar el uso del término objeto dentro de la tesis ya que lo utilizo de maneras distintas. La conjunción de actividades artísticas, tecnológicas e

investigativas que articulan al proyecto SonoTexto hace énfasis en el término *objeto* que, desde varias acepciones, lo coloca como objeto de estudio, tecnología y salida artística. Así, por *objeto computacional* me refiero a una pieza de software, sea esta un lenguaje de programación, un programa o un bloque de código fuente que extiende lo anterior independientemente del paradigma de programación empleado. La palabra *objeto* la uso, de manera específica, en el contexto del paradigma de la programación orientada a objetos para referirme a la instancia que se desprende de una *clase*. Y, aunque no utilizo el término *objeto de arte*, sea este en su acepción de obra musical o de producto artístico finalizado, es pertinente hacer la aclaración por la atracción que hay por definir el resultado estético de la investigación bajo este término y por la connotación de representación del paradigma OOP.

El término *objeto de arte* resulta problemático para la temática de este trabajo que se enfoca en el proceso de la práctica artística y no en el resultado finalizado derivado de ella. Esto puede sonar un tanto contradictorio con lo que se ha venido exponiendo pues para realizar live coding se requieren objetos computacionales "terminados". Sin embargo, tal terminación se encuentra sujeta a un trabajo en proceso que permite la reescritura del objeto computacional. Ello se entiende desde el proceso de las etapas técnicas de prueba y mantenimiento del ciclo del desarrollo de software y desde la constante exploración de la práctica artística como creación musical basada en la improvisación.

Para complementar lo anterior, el objeto computacional y la práctica artística de SonoTexto difieren de la idea de *objeto de arte* criticada por Christopher Small (1998), quien relaciona este término con una *obra* terminada, autocontenida, significativa por sí misma y desligada de su contexto. Esta idea, según el mismo autor, fue construida en el siglo XX, la que asigna a la música y al arte, vistos como *objetos*, un mayor valor ante la acción social de la música. En el caso de esta investigación, la práctica artística vinculada a su objeto computacional se define mejor bajo las características que Magnusson (2019) propone para las prácticas musicales del siglo XXI: evanescente, situada, dependiente de la tecnología y de su contexto.

Lo anterior sugiere un traslado del *ethos* de la práctica de live coding, que parte de la exploración, a lo que Haworth (en Dean y McLean, 2018) llama la negociación de lo abierto-terminado y Serdá (2005), en el contexto de la improvisación con computadoras, lo define como

un trabajo en permanente proceso que implica una relación entre imaginación, diseño, manufactura de computadoras musicales e improvisación. Tales conceptos resuenan con el término de las ciencias de la computación *beta perpetuo* que se refiere al software en mantenimiento constante y de manera indefinida durante su proceso de desarrollo[53].

En este contexto, tanto el objeto computacional como la práctica artística se realizan con la escritura de código fuente o programación, sin embargo, lo hacen desde aproximaciones diferentes. Por un lado, la práctica artística se lleva a cabo con la programación exploratoria u holística en el momento de realizar live coding y, por otro, el objeto computacional se desarrolla con la programación prescriptiva de su manufactura. De esta manera, SonoTexto es una práctica artística que actúa en interdependencia con su objeto computacional al aproximarse desde las dos formas de programación. Es decir, el código prescriptivo que construye al objeto computacional opera entrelazado a la escritura exploratoria de código fuente que genera la práctica artística en el momento de una presentación.

A su vez, SonoTexto se sitúa en un contexto relacional y colaborativo ya que, por un lado, captura el sonido del entorno o situación donde se lleva a cabo la presentación, y por otro, contiene conocimiento técnico compartido en foros, reuniones, tutoriales, repositorios y manuales escritos de manera colectiva. Asimismo, tanto el objeto computacional como la práctica artística están influenciados por el primer live coding practicado en México en su modalidad *from scratch* y por el pensamiento de apertura del código fuente que muestra cómo está programado, lo que supone un acceso para construir objetos computacionales, prácticas artísticas y discursos. De esta manera, el desarrollo tecnológico, el conocimiento colectivo y la postura por lo abierto delinean la forma de expresión de la práctica artística que, junto a los aspectos técnicos y estéticos incorpora los de carácter social.

Lo que expone esta sección reconoce al objeto computacional como el desarrollo tecnológico donde se implementan las ideas de la práctica artística. A esto me refiero en mi planteamiento cuando propongo diseñar un objeto computacional como parte de la práctica artística. Esto, además, se relaciona con los temas de apertura y abstracción, desarrollados en el

53 Para una descripción más amplia del término *beta perpetuo* se puede consultar la siguiente entrada de Wikipedia: https://es.wikipedia.org/wiki/Beta_perpetuo.

capítulo anterior, que permitirán mostrar cómo se transponen los conocimientos de la práctica artística y el desarrollo tecnológico en el caso de SonoTexto. Esto permitirá entender qué de la práctica artística es diferente cuando parte del desarrollo de un objeto computacional propio.

Lo anterior permite contestar *a priori* la interrogante sobre lo que posibilita musicalmente el desarrollo de un objeto computacional que parte de la práctica artística. En el caso particular de SonoTexto, la posibilidad de grabar sonido y organizarlo en el momento de la presentación apunta a un tipo música algorítmica improvisada con sonido grabado. Esto, como veremos a lo largo de las siguiente secciones, involucra una relación más compleja que aquella de la creación artística y la tecnología.

## 4.2 El desarrollo tecnológico de SonoTexto y su relación con la apertura y los niveles de abstracción

Cuando me enfrenté a “entrar al software” encontré que la programación exploratoria a la que estoy habituado es distinta a la programación prescriptiva empelada en el desarrollo de software. Ambas lidian con código fuente y algoritmos, su sintaxis es parecida, pero sus estructuras y forma de organización son diferentes. Desde esta perspectiva, la entrada al software tiene dos formas de acceso: usándolo o desarrollándolo. El discurso del live coding que propone “borrar” la barrera entre artista y programador (Baalman 2015; Jack, 2019; McLean 2011) ocurre en la programación exploratoria pero no del todo en la programación prescriptiva. Si bien el código fuente del software puede estar disponible, el acceso al entendimiento de sus estructuras para estudiarlo, modificarlo y eventualmente desarrollarlo depende de un conocimiento especializado.

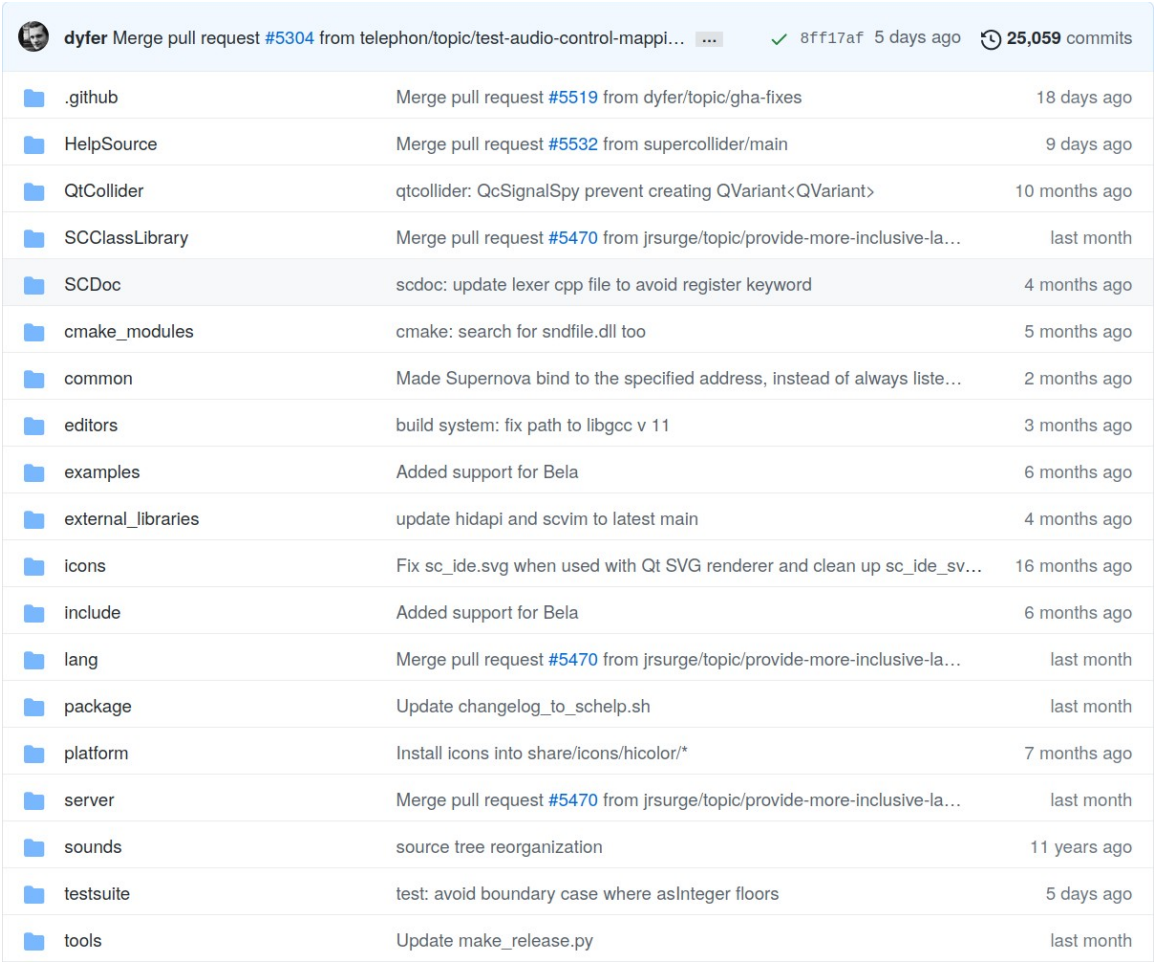



*Figura 4.3. Captura de pantalla de las carpetas que contienen el código fuente de SuperCollider en el repositorio de la plataforma GitHub. Tomada el 7 de agosto de 2021.*

Al ingresar al repositorio que contiene el código fuente de SuperCollider vemos una serie de carpetas y archivos con diferentes extensiones que contienen el código fuente del programa sin compilar, ejemplos, documentación e información de licencias y versiones[54]. Al navegar dentro de una de las carpetas, como el caso de *lang*, accesamos a otras carpetas que contienen diversos archivos, los cuales constituyen parte de las diferentes piezas de código necesarias para compilar el programa. Estos archivos, en específico, están escritos con el lenguaje de programación C++. Luego, hay otras carpetas como la que contiene la librería de clases SCClassLibrary que están escritas con el propio lenguaje de SuperCollider llamado sclang o los

54 El código fuente de SuperCollider se puede consultar en: https://github.com/supercollider/supercollider.

archivos de ayuda HelpSource programadas con el lenguaje de marcación de SuperCollider llamado SCDoc. Asimismo, algunos otros archivos están programados con los lenguajes de programación Python y JavaScript. Una vez compilado e instalado el programa, este se divide en tres partes: una que genera el sonido o *scsynth*, otra que controla cómo se genera el sonido o *sclang* y una que muestra la interfaz de usuario o *scide*.

Además de los diferentes archivos y carpetas mostradas en la página de inicio, podemos observar los *commits* o modificaciones realizadas y la metaestructura del programa (a través de la organización de sus carpetas). Esto nos muestra que el programa se compone por diversos archivos de texto que lo describen y no solo por el código fuente.

Más aún, programar de forma prescriptiva no garantiza el conocimiento para modificar el software de código abierto al que se accesa pues se requiere conocer el lenguaje de programación con el que está escrito. Por ejemplo, Hydra está programado con JavaScript, Sonic Pi con Ruby, C++ y Erlang, TidalCycles con Haskell y SuperCollider con C++. Por otro lado, aún conociendo el lenguaje de programación de desarrollo de un software resulta difícil leer el código fuente si no tiene comentarios escritos por los desarrolladores. Este embrollo se debe a que las tecnologías de live coding, como son los lenguajes de programación para realizar esta práctica, están construidas encima de otros lenguajes de programación. El desarrollo tecnológico no comienza desde cero, pues lo que ya está construido embebe a nuevas construcciones dentro de su tecnología. Esto lo menciona Wakefield y Roberts (2017) cuando dan cuenta del creciente número de lenguajes de programación de dominio específico altamente idiosincráticos en el ámbito del live coding. Los autores mencionan los casos de TidalCycles que catalogan como una especialización de dominio específico de Haskell o ixi-lang[55], el minilenguaje escrito sobre SuperCollider que, a su vez, dicen, es una especialización de dominio específico de SmallTalk.

Al observar y estudiar cómo están organizados el código fuente y los archivos dentro del repositorio de SuperCollider, me dí cuenta que no buscaba modificarlo en su nivel bajo programado en C++, sino diseñar una “forma de expresión” en el nivel alto escrito con sclang. De manera precisa, la estructura que requería diseñar era una *clase* como las que se encuentran en la librería del programa SCClassLibrary. De esta manera, me pareció conveniente estudiar la

55 Repositorio de ixi-lang: https://github.com/thormagnusson/ixilang.

estructura de tal elemento dentro de la estructura y organización del desarrollo de SuperCollider. Ello me posibilitaría concretar la idea surgida durante la práctica artística en una clase y conectarla al bucle de la investigación y desarrollo tecnológico.

Dentro del paradigma OOP, al que pertenece SuperCollider, una *clase* es un elemento que describe los datos y funciones para formar un objeto. Es decir, la *clase* es una especie de molde o plantilla de la que se desprenden réplicas llamadas objetos o instancias. En programación el término instancia es sinónimo de objeto, esto lo explica la sección *Introduction to objects* de SuperCollider cuando menciona que “todos los objetos son instancias de alguna clase, la cual describe la estructura del objeto y sus operaciones” (SuperCollider, 2021). Por su parte, Brookshear (2012), dice que un objeto es una unidad autocontenida que encapsula propiedades descritas en una plantilla llamada clase.

Con base en lo expuesto, las razones que me llevaron a materializar mi idea en una clase de SuperCollider fueron: 1) la clase es un elemento del lenguaje de programación que permite diseñar comportamientos, 2) SuperCollider es uno de los programas más utilizados en el live coding, 3) el código fuente del programa es abierto, 4) cuenta con una amplia comunidad, 5) es utilizado en la estructura que corre debajo de otros programas de live coding musical como TidalCycles, FoxDot y Sonic Pi y, 6) es soportado por computadoras de una sola placa como Raspberry Pi y Beaglebone. Lo anterior me permitió acceder al conocimiento generado sobre un software de amplia difusión y dejó abierta la posibilidad de conectar las clases desarrolladas con otros lenguajes de programación y plataformas hardware.

Pero, en qué momento hablamos de clase y en qué momento de objeto. Hay que aclarar que en programación orientada a objetos la clase no es lo mismo que el objeto. Formalmente una clase construye a un objeto, el que llevará su nombre y sus características. Bajo esta lógica, la clase puede verse como el contenedor de código fuente “estático” que describe las acciones, mientras que el objeto como una “copia activa” de la clase que media la traducción y ejecución de las acciones con la computadora.

```
Clase {
	variable
	método { }
	}
```

*Figura 4.4. Estructura general de una clase en SuperCollider.*

De manera general, una clase en SuperCollider (véase *Figura 4.4*) describe la estructura e implementación de un objeto y cuenta con los siguientes elementos: definición de clase, variables, y métodos. La definición de clase asigna un nombre único a la clase y comienza con una letra mayúscula. Después, se abre un par de llaves que aloja a los diferentes elementos de la clase. En la parte superior se declaran las variables, cuya función es asignar valores y guardar datos que serán usados dentro de la clase y por el objeto. Posteriormente se encuentran los métodos que contienen las funciones o procedimientos que realiza la clase, los que en el momento de crear el objeto son tratados como mensajes.

Las clases incluidas en el programa SuperCollider se encuentran en una carpeta especial llamada *SCClassLibrary* mientras que las clases desarrolladas por nuestra cuenta se colocan en la carpeta llamada *Extensions*. La *Figura 4.5* muestra la estructura del árbol del directorio de una clase hipotética externa colocada en la ruta de *Extensions*. La carpeta se identifica con el mismo nombre de la clase. Adentro de la carpeta se encuentra un archivo con extensión .sc que es el que reconoce el programa propiamente como la clase. De manera opcional se pueden colocar ejemplos del uso de la clase con la extensión .scd, los que deben abrirse dentro de la IDE del programa y declararse manualmente. Por último, hay una carpeta llamada HelpSource que contiene la carpeta *Classes* y adentro se encuentra el archivo de ayuda con extensión .schelp. Este archivo está escrito con el lenguaje de marcación de SuperCollider y por convención se guarda en la ruta de carpetas mostrada.

```
SuperCollider
    └── Extensions
        └── MiClase
            ├── miclase.sc
            ├── ejemplos.scd
            └── HelpSource
                └── Classes
                    └── MiClase.schelp
```

*Figura 4.5. Estructura de los archivos que conforman una clase externa en el programa SuperCollider.*

Al abrir SuperCollider, el programa compila diferentes directorios entre los que se encuentran aquellos que contienen las clases. Para ello busca la ruta de la carpeta SCClassLibrary que contiene las clases del programa. Luego busca la ruta de la carpeta Extensions y en caso de encontrar archivos con la extensión .sc los compila junto a las clases del programa. De esta manera, si la clase que diseñamos no contiene errores, esta es reconocida en el proceso de compilación cuando arrancamos el programa. Una vez iniciado el programa, la clase podrá ser "convertida" en objeto y, de esta manera, ser "programada" con el lenguaje sclang; asimismo, podrá mostrar la documentación contenida en sus archivos de ayuda. El proceso consiste en escribir el nombre de la clase, o asignarlo a una variable, en el IDE del programa y declarar (presionar una combinación de teclas sobre el código) la línea de código para generar el objeto.

Con base en lo anterior, escribí los objetos computacionales que conforman el proyecto SonoTexto, los que consisten en tres clases de SuperCollider: 1) *SonoTexto* para grabar fragmentos de sonido durante una presentación de live coding, 2) *SampleTexto* para reproducir sonidos almacenados en el disco duro y 3) *Ptexto* para secuenciar los sonidos de una lista con la operación matemática módulo.

*SonoTexto*

La clase SonoTexto graba fragmentos de sonido en cuatro *buffers* temporales.

Los buffers 1 y 3 graban sonido monoaural de 5 y 10 segundos de duración respectivamente y los buffers 2 y 4 graban sonido estéreo igualmente con duración de 5 y 10 segundos.

| **Nombre del Buffer** | **Duración** | **Número de canales** |
|---|---|---|
| \b1 | 5 segundos | monoaural (1) |
| \b2 | 5 segundos | estéreo (2) |
| \b3 | 10 segundos | monoaural (1) |
| \b4 | 10 segundos | estéreo (2) |

*Tabla 4.1. Duración y número de canales de los buffers de la clase SonoTexto.*

```
SonoTexto {
      boot {verificar la carpeta de sonidos y llamar los archivos
necesarios}
      rec {si 1, 2, 3 o 4 son verdaderos grabar e imprimir “grabando”, si
son falsos no hacer nada}
      write {si 1,2,3 o 4 son verdaderos escribir en el disco duro el
sonido almacenado en el buffer correspondiente, asignar un nombre e
imprimir “escribiendo”, si son falsas no hacer nada}
      read {leer los archivos de sonido guardados en la carpeta
<sonotexto> según su índice dentro de esta}
      info {informar cuántos sonidos contiene la carpeta <sonotexto>}
   }
```

*Figura 4.6. Pseudocódigo de la clase SonoTexto.*

La clase consta de los siguientes métodos: `boot`, `rec`, `write`, `read` e `info` (véase § Anexo A.1). El método `boot` prepara al objeto para grabar, para ello llama al archivo sonotexto-synths.scd, el cual asigna los cuatro buffers de grabación y reproducción a la memoria temporal de la computadora. Asimismo, este método verifica si existe la carpeta <sonotexto> donde se guardan los archivos de sonido que se quieren conservar. El método `rec` graba sonido en uno o varios de los cuatro buffers que se indiquen. El método `write` escribe en el disco duro los sonidos almacenados en el buffer o los buffers que se indiquen, el método `read` lee los archivos

guardados en la carpeta <sonotexto> y, finalmente, el método `info` informa cuántos sonidos hay en dicha carpeta.

La estructura de la clase está conformada por cuatro archivos: sonotexto.sc, sonotexto-synths.scd, sonotexto-examples.scd y sonotexto.schelp. El primer archivo contiene el código fuente de la clase con los métodos que llaman a las funciones para realizar las acciones programadas. El segundo archivo, con extensión scd, contiene el código fuente que asigna cuatro buffers, cuatro sintetizadores de grabación y cuatro sintetizadores de lectura. El tercero, también con extensión scd, contiene ejemplos del uso de la clase. El cuarto, con extensión schelp, es el archivo de ayuda de la clase en formato de marcación SCDoc colocado dentro de la carpeta *HelpSource* que permite ver su contenido dentro del navegador de ayuda del IDE del programa. Además, es necesario crear manualmente la carpeta <sonotexto> en el directorio *Recordings* del programa para almacenar los sonidos cuando se utiliza el método `write`. Aparte de los archivos mencionados, cuando la carpeta de la clase se sube al repositorio GitHub, se añaden los README.md y LICENSE. El primero contiene la información que se despliega en el sitio web del repositorio y el segundo la información sobre la licencia utilizada.

```
└── SuperCollider
            ├── Extensions
            │   └── sonotexto
            │         ├── HelpSource
            │         │   └── Classes
            │         │       └── SonoTexto.schelp
            │         ├── sonotexto.sc
            │         ├── sonotexto-synths.scd
            │         └── tests
            │             └── sonotexto-examples.scd
            └── Recordings
                └── sonotexto
                    └── 0000-b1-190529_132211.wav
```

*Figura 4.7. Estructura del árbol de archivos de la clase SonoTexto dentro de SuperCollider.*

La clase SonoTexto pasa por diversos procesos desde que abrimos el programa SuperCollider hasta que se declara la clase para formar el objeto y accionar la ejecución de las

tareas programadas. Primero, al abrir el programa hay una compilación de las clases y quarks que están en las rutas que reconoce el programa, proceso que es informado en la ventana *post*. Luego, ya con el programa abierto como muestra la *Figura 4.8*, se encuentran la líneas de código necesarias para crear el objeto SonoTexto y programar con él dentro de SuperCollider. Para crear el objeto y dejarlo listo para grabar se declaran las primeras tres líneas en este orden 1) la línea `s.boot` arrancar el servidor de SuperCollider, 2) la línea `t=SonoTexto` crea el objeto dentro de la variable `t`, 3) la línea t.boot habilita al objeto SonoTexto para grabar.

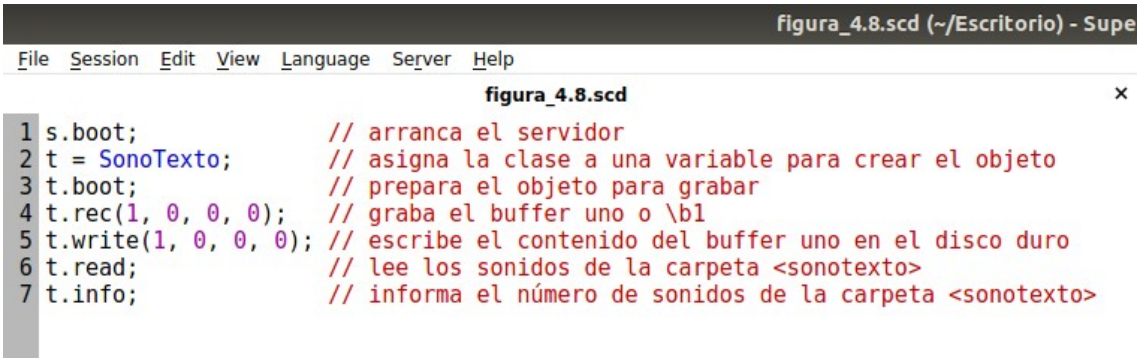



*Figura 4.8. Código comentado para programar con SonoTexto.*

Luego continúan las líneas que permiten operar las funciones del objeto recién creado. Para ello declaramos lo siguiente: 4) la línea `t.rec(1, 0, 0, 0)` le dice al objeto que grabe, o aloje, en la memoria volátil la representación numérica del sonido, en este ejemplo lo hace en el *buffer* 1, 5) si corremos la línea `t.write(1, 0, 0, 0)` la información almacenada en el buffer 1 se guardará como archivo WAV en la carpeta <sonotexto> con un identificador numérico que contiene la fecha y hora, 6) la línea `t.read` activa la lectura de los archivos WAV guardados en la carpeta <sonotexto> y, 7) la línea `t.info` imprime en la ventana *post* cuántos archivos WAV están almacenados en la carpeta.

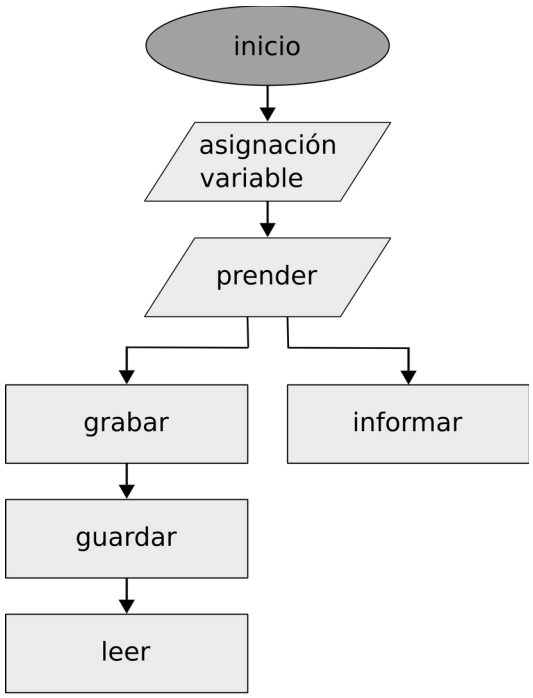



*Figura 4.9. Flujo de señal de SonoTexto.*

La idea de la clase, más que accesar a sonidos previamente grabados y almacenados en una carpeta, es la de reproducir y modificar de manera inmediata el sonido volátil alojado en los buffers que acaba de ser grabado. Para ello, hay cuatro sintetizadores que permiten tal acceso. Los parámetros de los sintetizadores están descritos en cada `SynthDef` de reproducción de los buffers del documento sonotexto-synths.scd (véase Anexo A.1). Tales parámetros son: `rate` (velocidad de lectura), `tg` (trigger), `st` (start position), `lp` (loop), `pb1` (panorama de buffer 1), `ab1` (amplitud del buffer 1), `atb1` (ataque de buffer 1), `sb1` (sostenimiento de buffer 1), `rb1` (relajamiento de buffer 1) y `ob1` (salida de buffer 1). La *Figura 4.10* muestra cómo pasar algunos parámetros al nodo `Synth` del buffer 1 o `\b1` para hacer sonar los sonidos recién grabados.

```
Synth(\b1, [\rate, 2, \atb1, 0.1, \sb1, 8, \rb1, 0.1])
```

*Figura 4.10. Línea de código para sonar el buffer 1 con algunos parámetros ordenados como dupla llave-valor.*

El nombre de los argumentos está inspirado en la mnemotecnia usada en los lenguajes de programación de bajo nivel. Estos se caracterizan por usar palabras reducidas que facilitan su memorización y escritura, lo que resulta conveniente para una presentación de live coding. El número que compone parte de cada mnemónico indica el buffer al que pertenece con excepción de los argumentos `rate`, `tg`, y `st` que usan nombres reservados por el programa para poder utilizar llaves especiales para la frecuencia. El sonido es controlado por eventos en pares formados por una llave y un valor. La llave indica el parámetro que se controla y el valor indica la cantidad. Esta dupla está separada por una coma como muestra la llave-valor del parámetro velocidad del ejemplo `\rate, 2`.

La clase SonoTexto se vincula al problema de investigación como el vehículo inicial de la transposición entre la programación exploratoria y la programación prescriptiva. Lo anterior se muestra, por un lado, en el código fuente exploratorio de la práctica artística expuesto en los ejemplos de las *Figuras 4.8* y *4.10* y, por otro, en el código fuente del desarrollo tecnológico expuesto en el Anexo A.1. El desplazamiento entre la práctica artística y el desarrollo tecnológico ocurre cuando el código fuente de la clase SonoTexto se convierte en un objeto que se puede programar de manera exploratoria durante una presentación. Esta conversión representa una escalada que va de la abstracción baja de la *clase* a la abstracción alta del *objeto*. Allí, el código prescriptivo de la clase permanece oculto y es mediante el código exploratorio del objeto que accesamos a la funciones encapsuladas. Aquí, el desarrollo tecnológico, o sea la clase, contiene la implementación de la idea musical, es decir, la serie de pasos necesarios para que pueda operar dentro de la computadora. La transformación de la clase en objeto durante la presentación trae consigo un cambio en la intención de programar que genera la posibilidad de que el pensamiento artístico impacte la estructura del software abierto.

*SampleTexto*

SampleTexto (véase § Anexo A.2) es una clase que lee archivos de sonido almacenados en una carpeta por número de índice o posición numérica y puede crear un `SynthDef` por cada uno de ellos. Para esto, la clase crea varios diccionarios que agrupan los sonidos por su cualidad

multicanal, sea esta monoaural o estéreo. Los métodos de la clase son `new`, `init`, `st`, `stsel`, `stm`, `sts`, `monosynth`, `stereosynth` e `info`.

```
SampleTexto {
    init {indicar la ruta de la carpeta de los archivos de sonido}
    st {leer los sonidos de la carpeta seleccionada}
    stsel {crear dos diccionarios:uno para sonidos monoaural y otro para
sonidos estéreo}
    stm {leer los sonidos del diccionario monoaural}
    sts {leer lo sonidos del diccionario estéreo}
    monosynth {crear un SynthDef por cada sonido mono definido por stsel}
    stereosynth {crear un SynthDef por cada sonido estéreo definido por
stsel}
    info {decir cuántos sonidos hay en la carpeta seleccionada}
}
```

*Figura 4.11. Pseudocódigo de la clase SampleTexto.*

El método `new`, conocido como *método constructor* (MDN, 2005) en la programación, es un método de clase heredado que crea e inicializa el objeto. Luego, el método `init` define la ruta de la carpeta donde se encuentran los archivos de sonido almacenados; `st` lee cualquier sonido de la carpeta; `stsel` crea un diccionario para los sonidos monoaural y otro para los sonidos estéreo; `stm` lee los sonidos monoaural y `sts` lee los sonidos estéreo; `monosynth` y `stereosynth` crean un `SynthDef` por cada archivo de sonido, mono o estéreo, a partir de los diccionarios creados por el método `stsel;` finalmente `info` indica el número de sonidos almacenados en la carpeta seleccionada.

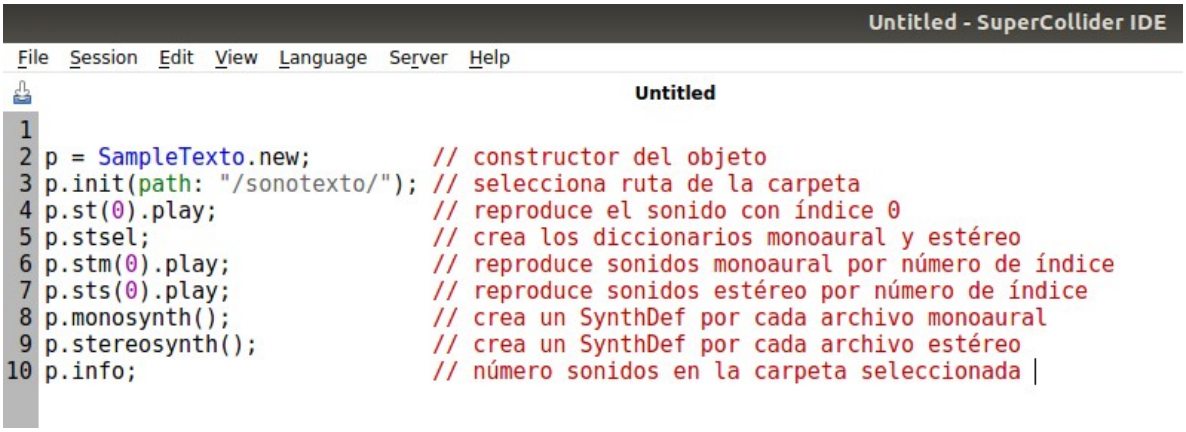



*Figura 4.12. Código de SampleTexto con métodos de la clase y métodos heredados.*

Esta clase, además de utilizar sus métodos, aprovecha la propiedad de *herencia* de los lenguajes de programación orientados a objetos para acceder a métodos de otras clases de mayor jerarquía llamadas superclases. Por ejemplo, hereda los métodos de instancia `play`, `plot`, y `normalize` de la clase Buffer, los que permiten leer, visualizar y normalizar con los métodos `st`, `stm` y `sts` el sonido sin la necesidad de volver a escribir esos métodos dentro de su estructura.

Cabe aclarar que los métodos `monosynth` y `stereosynth` fueron implementados posteriormente para aprovechar las posibilidades de manipulación del `SynthDef`, esto a diferencia de la lectura directa del archivo generado por el diccionario. Los ejemplos mostrados en la siguiente sección, que utilizan esta clase, se realizaron con los archivos generados directamente por el diccionario (véase *Figura 4.22* y *Figura 4.23*). Esto, como veremos más adelante, implica comenzar las presentaciones con código preparado en lugar de partir desde cero.

El mecanismo de organización y lectura de archivos guardados en el disco duro de SampleTexto puede pensarse como un módulo que expande las posibilidades de la clase SonoTexto. La idea de su diseño surgió en el momento que la práctica artística enfrentó el cambio de “escenario” debido a la pandemia. Esto ocurrió cuando la grabación de sonido durante el tiempo acotado de la presentación pública pasó a la captura de sonidos en la temporalidad de los días del confinamiento. Durante ese periodo, utilicé SonoTexto para grabar y guardar en el disco duro los sonidos de mi entorno que llamaban mi atención. Esta dinámica dio la pauta para

diseñar una clase que pudiera leer los archivos de sonido almacenados en el disco duro con el fin de utilizarlos durante algunas presentaciones en línea como veremos en ejemplos posteriores.

Si bien estas circunstancias transformaron la práctica artística planteada en la propuesta de investigación, el desarrollo de la clase SampleTexto se dio bajo la misma metodología que la clase anterior. En este sentido, cuando utilizamos la clase SampleTexto se activa la transposición entre las capas de la práctica artística y el desarrollo tecnológico posibilitadas por la apertura y la abstracción de SuperCollider. Esto es, las funciones para llamar sonidos del disco duro quedaron implementadas en una clase que, en el momento de realizar live coding *from scratch*, no solo resuelven la lectura de archivos sonoros sino que forman parte de la estética visual del código proyectado que muestra pocas líneas gracias a la abstracción.

*Ptexto*

La clase Ptexto (véase § Anexo A.3) es un patrón para organizar el sonido en el tiempo con el operador módulo o resto de una división. Este operador se expresa en SuperCollider con el símbolo % o el mensaje `mod`. Por ejemplo, la división 17 entre 8 tiene un residuo de 1 y se puede expresar de dos formas: `17%8` o `17.mod(8)`. La clase del patrón Ptexto comenzó su desarrollo por dos razones: primero para entender cómo están escritas las clases de la librería de patrones de SuperCollider y segundo para encapsular una línea de código que suelo utilizar de manera recurrente cuando realizo live coding.

La línea de código en cuestión es `Pseries(1, 1, inf) % 1.5 + 0.5`. La idea de esta línea es aplicar la operación módulo a cada uno de los valores de una serie de números y colocar el resultado en un parámetro musical. Inicialmente la utilicé para generar ritmos, para ello apliqué los valores resultantes de la operación a la duración de un `Pbind` como se observa en la *Figura 4.13*.

```
Pbind(
    \instrument, \default,
    \dur, Pseries(0.5, 1, inf) % 1.5 + 0.5
    ).play
```

*Figura 4.13. Línea "goto" de la que parte el diseño de Ptexto.*

Lo que expresa el ejemplo es una secuencia de duraciones para crear un ritmo con el instrumento genérico de SuperCollider. La información que recibe la llave de duración `\dur` es una serie de números en secuencia proveídos por la línea de código en discusión. En el ejemplo, la serie de números dentro del patrón `Pseries` comienza con el valor 0.5 y se va incrementando gradualmente al infinito por valores de 1 para generar la secuencia 0.5, 1.5, 2.5, 3.5, 4.5 … n. Cada valor arrojado es dividido por 1.5 para obtener la secuencia formada por los residuos de la operación módulo 0.5, 0, 1, 0.5, 0, 1 … n. Posteriormente, a cada valor se le suma 0.5 para evitar que el residuo 0 pase a la duración y colapse el programa por un valor nulo. El resultado final es la serie 1, 0.5, 1.5, 1, 0.5, 1.5 … n. Estos últimos valores son los que pasan en forma serial a la llave de duración para formar el ritmo.

```
inicio
   generar serie desde 0.5 en incrementos de 1 < 0.5, 1.5, 2.5, 3.5, 4.5 … n
   aplicar módulo 1.5 a la serie generada < 0.5, 0,  1, 0.5, 0 … n
   sumar 0.5 a la serie pasada por módulo < 1, 0.5 1.5, 1, 0.5 … n
   escribir el resultado en duración < 1, 0.5 1.5, 1, 0.5 … n
fin
```

*Figura 4.14. Pseudocódigo de Ptexto.*

La secuencia final aplicada a la duración genera un ritmo que podría colocarse directamente con el patrón de secuencia `Pseq([1, 0.5, 1.5], inf)`. Sin embargo, llegar a esa serie es resultado de un proceso de razonamiento distinto al de combinar valores de duración de forma secuencial. La forma en que llegué a esa línea fue probando distintos operadores binarios

aplicados a un patrón. En ese proceso me di cuenta de que el operador módulo arrojaba combinaciones rítmicas particulares. Con el tiempo fui refinando esta técnica y la incorporé a los recursos que tengo para improvisar con código, asimismo comencé a aplicarla a parámetros distintos a la duración como la altura y a combinarla con otros patrones[56].

Koffi Oduro, en la sesión de trabajo *Journaling* de Hybrid Live Coding Interfaces 2021, menciona la idea de recurrir a una línea de código que tenemos bien estudiada durante una presentación de live coding. Este artista digital y programador la llama "línea *GOTO"* en referencia al polémico comando de programación que sirve para forzar el salto a una línea de código específica dentro de un programa[57]. Oduro no la usa en ese sentido, más bien la toma como metáfora para plantear una pregunta a quienes participaban en la sesión. La formulación la hace de esta manera: "todo mundo tiene una línea de código favorita que si estuviera en un apuro, o algo parecido, la escribiría […] ¿cuál es tu línea *goto* en tu lenguaje?" (Oduro en Hybrid Live Coding Interfaces, 2021, 1:24:19). En mi caso, la línea *goto* que he utilizado durante algún tiempo es aquella que aplica el operador módulo como lo expresa el ejemplo antes mencionado.

Los retos que planteó la programación de Ptexto fueron varios. El primero fue estudiar y entender el mecanismo encapsulado dentro del método `embedInStream`, el cual coloca los eventos sonoros en el flujo temporal de los patrones. El segundo fue pensar en la acción o comportamiento que podía implementar para desarrollar un patrón y así probar tal mecanismo. Una vez que entendí la estructura del patrón para colocar eventos sonoros en el tiempo, decidí implementar mi línea *goto*, así que el tercer reto consistió en implementar esa línea en la estructura del patrón Ptexto.

Si bien el contenido de Ptexto se resuelve con la línea de código expuesta en la *Figura 4.13*, encapsularla en un patrón me motivó a programar de manera prescriptiva una línea de código "descubierta" en el proceso de programación exploratoria. Lo anterior es problemático, pues complejiza y encapsula la solución resuelta en la programación exploratoria con los recursos del lenguaje de programación. Por otro lado, el nombre Ptexto (Patrón + texto) no hace

56 En este ejemplo se puede observar, a partir del minuto 1:00, el uso de módulo % con la línea de código mencionada: https://youtu.be/Is49DCL4Kw8?t=60.

57 En este enlace se puede leer sobre la instrucción GOTO: https://es.wikipedia.org/wiki/GOTO.

referencia al operador módulo sino al juego de palabras de la clase SonoTexto, lo que lo hace significativo para mí pero no para una mirada externa.

```
Ptexto([2, 3, 4, 5], 1.5, 0, inf);
```

*Figura 4.15. Ejemplo de Ptexto en el que cada número de la lista es dividido por 1.5 para obtener el residuo.*

El patrón Ptexto está diseñado para introducir cuatro argumentos: `list` o lista de valores, `modVal` o valor del módulo, `sumVal` o valor de la suma y `repeats` o número de repeticiones que realizará el patrón. Tales argumentos fueron implementados con el método constructor `new` que permite ingresar estos valores al mecanismo `embedInStream`.

El ejemplo de la *Figura 4.15* se puede leer de izquierda a derecha. Primero se encuentra el nombre del patrón Ptexto. Luego, entre paréntesis, los valores asignados a los argumentos del método constructor `*new` (list, modVal, sumVal y repeats). El primer argumento es la lista o arreglo de los números 2, 3, 4, 5 colocados entre corchetes. Luego, el número 1.5 que funge como divisor de la operación módulo. A continuación un valor que suma 0 al resultado de la operación módulo. Finalmente, el número de veces que el patrón realizará las operaciones descritas de manera secuencial, en esta caso expresando una cantidad de repeticiones infinitas.

El caso de Ptexto muestra un proceso más interiorizado de la transposición entre las capas y niveles de la práctica artística y el desarrollo tecnológico de las clases. Acá la transposición entre la capa de la práctica artística que parte de la línea *goto* y la capa de desarrollo tecnológico que se encuentra en el mecanismo del patrón, abre el camino de la transposición para navegar en la abstracción del lenguaje de programación de SuperCollider. La línea *goto*, surgida de la exploración del lenguaje, queda implementada en la clase-patrón Ptexto. Esa línea no es una idea musical abstracta, es una línea que implementa una posibilidad rítmica que surgió de la dinámica de prueba y error en el proceso de programación exploratoria que, a su vez, quedó implementada en la estructura formal del patrón.

Para finalizar esta sección, las clases SonoTexto, SampleTexto y Ptexto fueron programadas bajo las convenciones de desarrollo del programa SuperCollider. Las decisiones de su desarrollo se informaron por la práctica artística y la investigación teórica. Asimismo, la situación provocada por el confinamiento impactó el cambio de énfasis sobre la grabación de sonido y el desarrollo de las clases subsecuentes. Las clases se encuentran disponibles en repositorios de la plataforma de desarrollo GitHub[58]. Cada repositorio contiene los archivos de código fuente de la clase, los archivos de ayuda, ejemplos y un archivo README en inglés y en español. El Anexo B de este trabajo contiene la información de cómo descargar e instalar las clases en SuperCollider. El código fuente se distribuye con una licencia abierta GNU *General Public License* v3.0 lo que da cuenta de su afiliación al software libre.

El encapsulamiento de código fuente en la abstracción de las clases favorece la circulación de las ideas musicales en el lenguaje de programación. De esta manera circulan los objetos computacionales que, aunque no son piezas musicales, tienen implementado un pensamiento musical. Con el desarrollo de estas clases la pregunta sobre la transformación que sufre la práctica artística en el desplazamiento hacia su desarrollo tecnológico se contesta parcialmente. Así, la práctica artística del live coding sistematiza el pensamiento musical en términos del desarrollo de software y, a su vez, los mecanismos exploratorios de la práctica artística pasan a una abstracción más baja.

## 4.3 Práctica artística con SonoTexto

El diseño inicial de la clase SonoTexto fue pensado para realizar live coding *from scratch* con sonidos del entorno acústico del lugar grabados en el momento de cada presentación. La clase contempla la improvisación con sonido capturado en la memoria temporal de la computadora que no existe antes de la presentación y que se borra cuando se apaga la computadora. Lo anterior hace referencia a la postura del manifiesto de live coding, expresada en el noveno punto,

58 https://github.com/hvillase/sonotexto.git, https://github.com/hvillase/sampletexto.git y https://github.com/hvillase/ptexto.git.

que aboga por no dejar registro o respaldo del código después de la presentación. Con ello me refiero a comenzar una presentación de live coding con la pantalla en blanco y mostrar a la audiencia el proceso de la escritura del código fuente, las decisiones de programación, de grabación y la organización del sonido. Tal interpretación del manifiesto viene de las Sesiones de Live Coding organizadas por el Cmm que instaban a quienes participaban a partir de cero, lo que empujaba el carácter improvisatorio de las sesiones.

La modalidad *from scratch* puede transmitir la idea de "virtuosismo" en la programación cuando se escribe desde cero todo el código durante la presentación. Sin embargo, esto es posible debido a diversos desarrollos informáticos previamente realizados y al cúmulo de experiencia y conocimiento de cada practicante. Esto es, en el momento de escribir código fuente en vivo "desde cero", lo hacemos sobre las soluciones del lenguaje de programación y desde el código fuente que hemos in-corporado en nuestra práctica.

La intención inicial de la práctica artística con la clase SonoTexto se manifiesta en los conciertos y presentaciones presenciales realizadas durante el año 2019 antes de la crisis sanitaria causada por la pandemia de COVID-19. A partir de ese suceso, que pasó poco antes de llegar a la mitad de la investigación, la práctica artística tuvo que adaptarse a las circunstancias del confinamiento que impedían continuar con eventos presenciales. Consecuentemente, el planteamiento, la intención y el desarrollo de la clase SonoTexto pasaron del uso en espacios públicos al uso en el espacio personal con las restricciones que ello implicó. A su vez, la presencia de público en un espacio físico se distribuyó geográficamente en una presencia mediada por los dispositivos interconectados a las plataformas donde se comenzaron a transmitir las presentaciones en línea.

Con base en lo anterior, he dividido los ejemplos de la práctica artística de esta sección en dos etapas: una presencial previa a la pandemia y otra en línea durante el transcurso de la crisis sanitaria. La primera etapa consiste en una serie de presentaciones realizadas en la fase inicial de la investigación durante el año 2019 y llegan hasta el inicio del año 2020, cuando es decretado el confinamiento como medida de contención. La segunda etapa transita desde el anuncio de dicha medida y se extiende al periodo de cierres, aperturas y semáforos de riesgo epidémico que coinciden con la fase intermedia de la investigación hasta su fase final. Durante estas etapas

realicé once presentaciones con las clases SonoTexto, de las que escogí algunos ejemplos que dan cuenta de la práctica artística. En el Anexo C he colocado todas las presentaciones en orden cronológico con un vínculo al registro audiovisual[59].

*Etapa presencial*

La primera etapa consiste en una serie de conciertos e improvisaciones presenciales llevadas a cabo en recintos con audiencia. El primer ejemplo seleccionado es el video de la improvisación realizada el 26 de junio de 2019 en la Galería el Rule en la Ciudad de México[60]. En esa ocasión el sonido del entorno acústico fue capturado en diferentes momentos de la presentación; el enfoque estaba en cómo resonaba el lugar, incluidos los sonidos que emitían las personas presentes en la sala, los sonidos que entraban del exterior y los emitidos por las piezas expuestas en las galerías anexas.

En ese momento la clase SonoTexto estaba en su primera etapa de desarrollo que consistía en registrar sonido en el momento. El método relevante de la clase era `rec` que activa el momento de inicio de la grabación de sonido en el `Buffer` seleccionado. El método `rec` está ligado a la escucha del entorno sonoro y a la decisión de captura de un fragmento de sonido. Una vez grabado el sonido en la memoria temporal de la computadora se da inicio a la escritura de código fuente para organizarlo en el tiempo. Este proceso lo realicé con las clases de SuperCollider llamadas Patterns o patrones, como se muestra en la *Figura 4.17*, lo cual es una forma común entre quienes practican live coding de organizar el código fuente en estructuras musicales con los programas SuperCollider, TidalCycles o Gibber. En el caso de SuperCollider, Magnusson (2021) se refiere a los patrones como formas eficientes de organizar estructuras musicales que controlan los objetos llamados `SynthDef` o sintetizadores. Menciono lo anterior pues los Buffers de SonoTexto que alojan temporalmente los sonidos grabados durante cada presentación están insertos en SynthDefs para poder modificarlos con diferentes patrones.

59 Esta lista contiene la mayoría de las presentaciones presenciales y en línea: https://youtube.com/playlist?list=PLFB-yISQMraUw9YtiBXH6vaEuA38oFYkN.

60 Presentación de live coding realizada en el Rule, ciudad de México https://youtu.be/nt3b20MfC70.


*Figura 4.16. Concierto de SonoTexto en Galería el Rule. Foto por Gabriel Sánchez.*

Los patrones en SuperCollider son clases que pertenecen a una librería diseñada para crear estructuras musicales. Estos describen el comportamiento de parámetros musicales y sonoros que ocurrirán en el tiempo. Son una especie de partitura que activa un flujo temporal y secuencial de valores que será "interpretada" con el instrumento digital `SynthDef`. La documentación de ayuda de SuperCollider define a los patrones como clases que "forman un lenguaje de notación rico y conciso para la música" (SuperCollider 3.12.0, 2021). Los patrones actúan junto a otros mecanismos para ejecutar las secuencias que describen; según el archivo de ayuda de `Pattern`, un patrón está definido por un `Stream` o flujo de valores que suceden uno a uno de manera secuencial y repetida. Además, los patrones pueden combinarse y anidarse dentro de otros patrones para generar secuencias de valores complejas aplicadas a diferentes parámetros musicales que queramos ordenar en el tiempo para enviarlos al sintetizador.

Una forma de mandar secuencias temporales de valores a los parámetros de un sintetizador es con el uso de la estructura `Pbind`. Tal estructura aloja en su interior duplas

compuestas por una *llave* y un *valor* que cumplen la función de dirigir los valores a los parámetros específicos de un sintetizador. Las llaves son palabras o mnemónicos antecedidas por una diagonal, por ejemplo `\dur`, que hacen referencia al nombre de un sintetizador, al de sus parámetros o a palabras reservadas por el programa. Los valores pueden ser números o secuencias de números producidos por un patrón. La dupla llave-valor separa sus elementos por una coma, por ejemplo `\amp, 0.5`. Un Pbind puede gestionar múltiples duplas para generar secuencias de valores que suceden de manera cíclica. Por su parte, el sintetizador recibe cada valor y lo va canalizando al parámetro cuyo nombre coincida con el de una llave.

```
t = SonoTexto;
t.boot;
t.rec(true, false, false, false);

Pdef(\rule,
   Pbind(\instrument, \b1,
  \dur, 4,
  \ab1, 2
)
).play
```

*Figura 4.17. Fragmento de código de la improvisación en la Galería del Rule.*

La *Figura 4.17* muestra un fragmento de código similar al programado en la presentación de la Galería del Rule. La primera línea de código fuente asigna la clase SonoTexto a una variable, la segunda activa la clase y la tercera graba sonido en algún buffer cuando el valor es verdadero. A continuación está la estructura `Pdef` o definición de patrón que sirve para modificar sobre la marcha el contenido del `Pbind` que se encuentra en su interior. La palabra `\rule` indica el nombre del `Pdef`. La estructura `Pbind`, que en este caso envuelve tres duplas de llave-valor, se activa en bucle cuando el `Pdef` recibe el mensaje `play`. En ese momento, el Pbind comienza a arrojar valores que causan que el sintetizador b1, el cual está encapsulado en la clase SonoTexto, suene cada 4 segundos con una intensidad de 2. A partir de entonces se pueden reescribir los

valores, añadir patrones, colocar más duplas de llave-valor y refrescar el código cada cierto tiempo para comenzar a escuchar las variaciones del sonido.

En el ámbito musical y sonoro lo que ocurre es que después de grabar el primer sonido, el live coder lo activa en loop y comienza a modificar sus parámetros. Luego es posible grabar un segundo, tercero y hasta un cuarto sonido y activar cada uno de ellos en loop para sumar una nueva capa de sonido a las que ya están sonando. De manera general, cada sonido puede variar su altura, su duración y su sentido de reproducción para formar texturas y ritmos. Como veremos más adelante, modificar cuatro capas de sonido durante una presentación de live coding con la escritura de código fuente es algo recurrente.

Las modificaciones del sonido causadas por la reescritura del código se incorporan de regreso al entorno acústico a través de las bocinas. De esta manera, la suma de ambas fuentes sonoras –sonido ambiente y sonido modificado en la computadora– sirve como referencia a la escucha para tomar la decisión de grabar otro fragmento de sonido o modificar el código del sonido, o sonidos, que está sonando en bucle. En ese momento la memoria, la escucha y la escritura de código son procesos activos. Aquí vale la metáfora del live coder como improvisador que “toca” un instrumento musical digital mediante un sistema de notación musical y computacional. Durante la presentación el live coder escribe secuencias de acciones que ejecuta el instrumento digital para producir sonido. Lo particular de esta interacción es que el live coder no acciona físicamente sobre el instrumento, lo hace a través de una dinámica de programación en la que primero escribe lo que imagina que podría sonar y luego delega al software de la computadora la tarea de traducir el código que escribió para que emita sonido.

A partir de este concierto comencé a realizar el registro de campo, tanto de video como de audio, para conformar los materiales de documentación para la tesis. Derivado de ello inicié una reflexión sobre cómo evaluar la experiencia del concierto presencial con relación al registro de campo al que nos enfrentamos de manera posterior. En este sentido, el registro en video no debe leerse como el resultado pues carece de los elementos acústicos de cada sitio y está grabado desde el encuadre subjetivo de quien coloca la cámara. En cuanto al registro de las presentaciones de live coding, Magnusson (2019) menciona que el medio computacional es distinto a otros medios de escritura, como el papel, donde prevalece el texto. En cambio, dice el

autor, la música algorítmica o generativa, a la que pertenece el live coding, se encuentra en un "estado constante de innovación y exploración" (Magnusson, 2019, p. 180). Lo anterior responde a lo que Magnusson llama nueva cultura oral del medio electrónico que, a pesar de ser documentada de manera audiovisual y diseminarse rápido, no se compone, consume y archiva concienzudamente. A pesar de que es excesivamente textual, dice que no hay un interés por archivar, más bien queda registrada en canales extratextuales como YouTube.

Este ejemplo muestra la relación entre la programación exploratoria y prescriptiva. Por un lado, el código fuente escrito durante la presentación y mostrado a la audiencia contiene, entre las líneas del código fuente exploratorio, los comandos que llaman al objeto computacional desarrollado de manera prescriptiva. Esta relación hace posible llevar a cabo una presentación que involucra la transposición entre el conocimiento contenido en las capas y niveles planteado en el capítulo anterior. Acá el conocimiento para utilizar la clase SonoTexto está ligado al conocimiento que permitió su desarrollo. Esto da cuenta de la transformación que sufre la práctica artística cuando las ideas que surgen de ella se implementan de forma sistematizada en el diseño de sus objetos computacionales.

El segundo ejemplo de conciertos presenciales es *Iterar el espacio*, ocurrido el 7 de noviembre de 2019 en la Facultad de Música de la UNAM dentro del ciclo Resonancias edición XXIX[61]. La dinámica del concierto fue similar a la anterior: escuchar el entorno, capturar el sonido y organizarlo con patrones, solo que en esa ocasión además utilicé cuatro bocinas para generar una experiencia envolvente. Antes del concierto, Jorge David García, quien coordina el ciclo Resonancias, me sugirió avisar a los asistentes que en algunos momentos grabaría sonido como una provocación para que emitieran sonidos. A raíz de esa experiencia agregué al código fuente de la clase el mensaje `"Recording n file"`, el cual aparece en la pantalla cuando comienza la grabación para que la audiencia lea dicho momento y emita sonido si lo desea. Esto se suma a la idea de grabar tres categorías de sonidos distintas que ocurren en el entorno de la presentación como es la participación del público.

61 Publicidad del concierto y su descripción https://www.facebook.com/events/2749420328456697/.

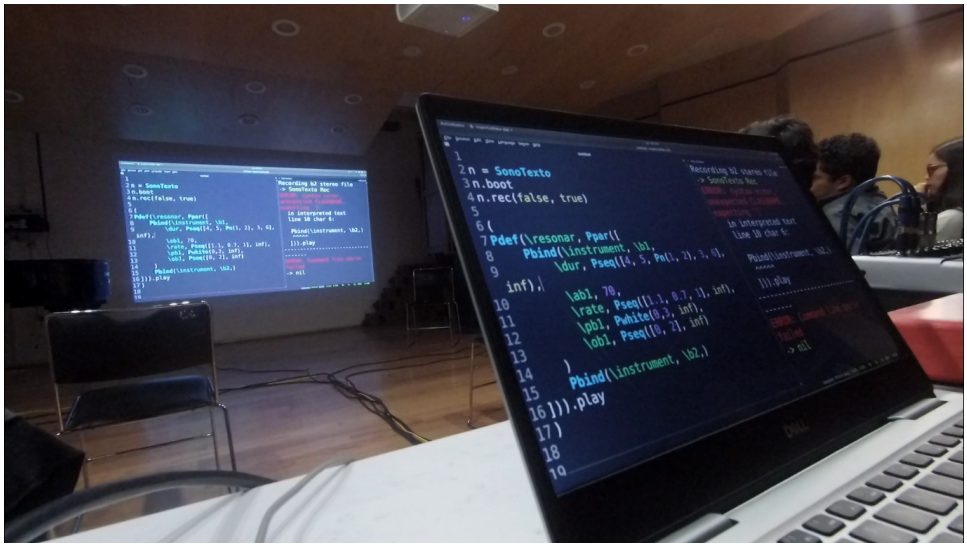


*Figura 4.18. Resonancias XXIX en Sala A 10 de la FaM UNAM, 7 de noviembre de 2019. Captura de pantalla del video de registro del autor.*

La reacción del público al entender que hay momentos en que se graba sonido ha consistido, en algunos casos, en hacer sonidos con la boca o con objetos. En el caso de la presentación de Resonancias resultó divertido para algunos asistentes leer en el código el momento de grabación para emitir o hacer algún sonido, lo que se observa en el minuto 15:24[62]. En presentaciones posteriores como la del ICLC 2020 y 8:08pm La Hora del Live Coder, que discuto más adelante, la gente pudo entender que estaba grabando sonido en algunos momentos, aunque se limitaron a comentar el hecho y no tanto a producir sonidos. Esto sugiere que no es suficiente colocar un mensaje del momento en que se graba para que el público participe, es necesario invitarlo a interactuar. No obstante la dinámica de este concierto, la investigación no se centró en buscar una interacción con el público como generador de sonido dentro del espacio sino en las posibilidades de capturar el sonido del espacio incluido aquel que produjera la audiencia.

El tercer ejemplo presencial ocurrió el 6 de febrero de 2020 en la serie de conciertos del V Congreso Internacional de Live Coding llevadas a cabo en el Cafe Allegro de la Sala de Conciertos de la Universidad de Limerick[63]. Aquella ocasión la improvisación se realizó con el

62 Registro de la presentación de Resonancias XXIX en archive.org: https://archive.org/details/resonancias-xxix.
63 Presentación en ICLC 2020 https://youtu.be/rejN3CHMbpw?t=4355.

sonido del entorno de la cafetería, el cual consistía en las voces de las personas y el sonido de cubiertos golpeando sobre platos y vajillas de un lugar concurrido a la hora de la comida.

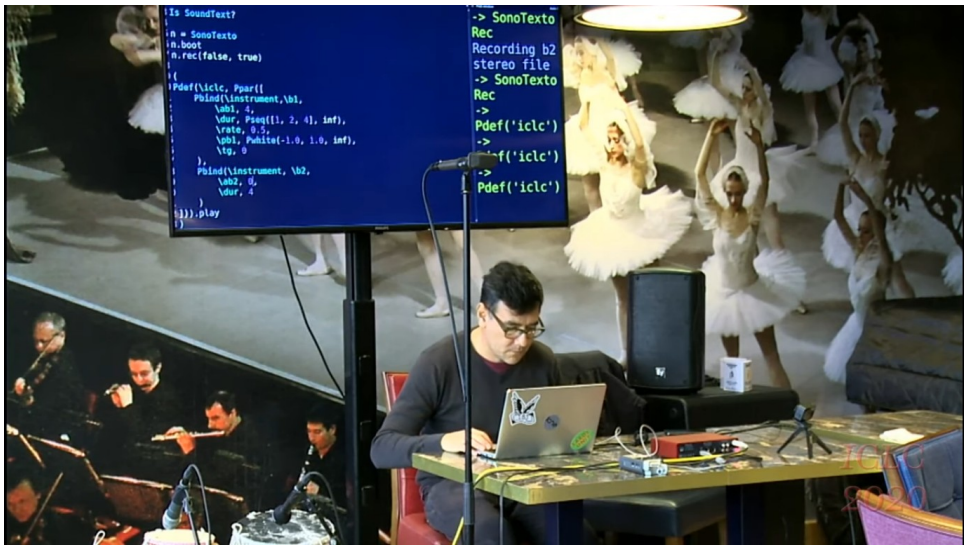


*Figura 4.19. SonoTexto: Cafe Allegro de la Sala de Conciertos UL, 6 de febrero de 2020. Captura de pantalla de YouTube ICLC.*

Escogí tocar en ese lugar para evitar una sala de concierto acondicionada que cancelara el sonido externo y la probabilidad de una audiencia lejana en el ámbito formal de la sala de conciertos. En cambio, la cafetería me proporcionó una gran cantidad de sonidos generados espontáneamente en el entorno. Algo que no había contemplado es que esta circunstancia de grabación puede capturar las conversaciones de las personas las cuales son amplificadas en el lugar. Esto puede irrumpir de manera sorpresiva en la audiencia si no hay un conocimiento previo de lo que sucederá en el lugar, lo que podría generar un problema de privacidad.

Lo anterior denota una situación más compleja que la simple interacción del live coder con el espacio y el sonido. Y es que, si bien la presentación de live coding responde al ritual del concierto musical, en este caso están en juego acciones que han sido objeto de reflexión en otros campos de práctica sonora. Particularmente me refiero al hecho de abrir un micrófono en un espacio público. Al respecto, Maja Zećo (2021), en el contexto de prácticas como la caminata

sonora y la grabación de campo, plantea que escuchar y grabar *in situ* nos relaciona de manera sociopolítica con el lugar dado que este es producido constantemente por la interacción de las personas y los agentes no humanos. En esta situación, quien escucha a través del micrófono capta vibraciones de la infraestructura, lo que para Zećo es una acción que produce conocimiento y de la cual debemos estar conscientes.

Por último, los ejemplos presentados en este apartado muestran la transposición entre las capas de la práctica artística y el desarrollo tecnológico a través de la programación exploratoria que actúa arriba de la programación prescriptiva de la clase SonoTexto. De manera explícita, el ejemplo de la Galería el Rule muestra el desplazamiento de la programación entre la capa de la práctica artística y la capa del desarrollo tecnológico. Ahí, las líneas de código 6 y 8 de la *Figura 4.17* dejan ver dos parámetros del objeto SonoTexto (\b1 y \ab1) que actúan entrelazados a las funciones del lenguaje de programación SuperCollider. El código del ejemplo muestra un momento de la programación exploratoria que pone en juego diferentes objetos del programa entre los que se encuentra SonoTexto. Las presentaciones de esta etapa dejan escuchar improvisaciones construidas con el sonido producido dentro de los espacios donde ocurrieron. Los sonidos del lugar se sobreponen conforme van siendo grabados y permanecen en bucles que generan masas sonoras de diferentes alturas a las que se incorporan sonidos incidentales como aquellos producidos por el público. Aquí el espacio es la fuente sonora que proporciona los sonidos que son moldeados durante cada presentación.

*Etapa en línea*

A inicios del año 2020 comenzó la transición al confinamiento en varios países como medida para frenar la pandemia de COVID-19 al que México entró en marzo de ese año. Esta etapa coincidió con el aniversario 16 de Toplap para lo que se organizó una transmisión en línea de 72 horas sin interrumpir. Este tipo de eventos en línea, independientemente del confinamiento, se realizan desde el año 2017 por medio de una convocatoria dirigida a quienes deseen transmitir

media hora de música programada en vivo o alguna acción relacionada con el live coding por el canal de YouTube de Eulerroom[64].

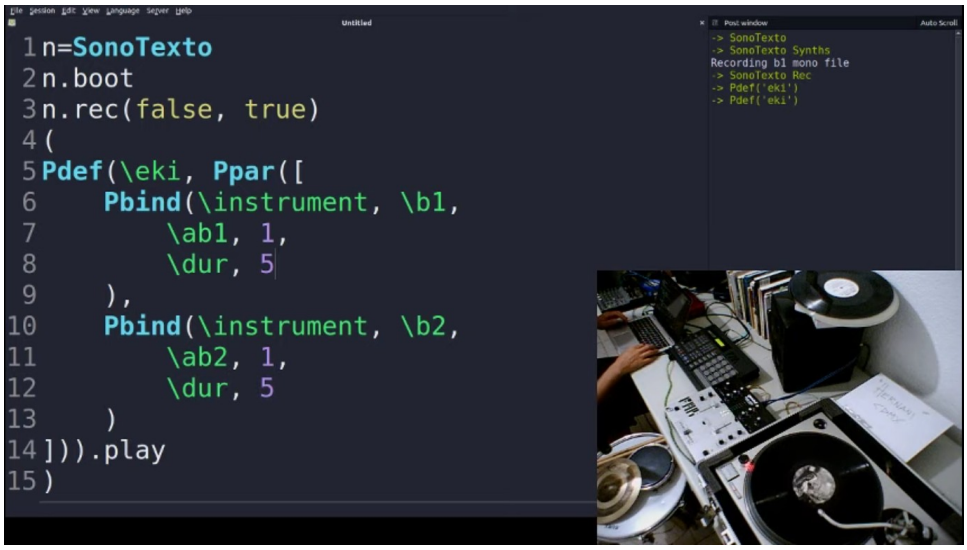



*Figura 4.20. Ensayo de Equinox Eulerroom 2020. Captura de pantalla del canal YouTube del autor.*

Para realizar las transmisiones circula un documento de Google con las instrucciones de transmisión con el programa *Open Broadcaster Software* (OBS). Dicho documento técnico, escrito para orientar a quien participa, ha sido distribuido, modificado y traducido a varios idiomas por algunas comunidades de live coding[65]. De esta manera, quienes hemos participado en los aniversarios o eventos en línea organizados por Toplap o Algorave aprendimos a transmitir audio y video con OBS. Aunque ya se realizaban este tipo de presentaciones antes de la pandemia, durante el confinamiento aumentaron y se reforzaron. Así, bajo el esquema de transmisión y organización usado por Toplap, varios colectivos de live coding comenzaron a realizar eventos de transmisión en línea. Esto trajo consigo una apertura en la participación dentro de las actividades de diferentes comunidades, lo que permitió comenzar a descentralizar

64 Wearefive fue un evento de transmisión en línea en marzo de 2017 donde se puede observar la organización de un concierto con 48 participantes de diferentes países https://algorave.com/wearefive/.

65 Véase la versión traducida al español del colectivo de live coding argentino CLiC: https://colectivo-de-livecoders.gitlab.io/blog/posts/2020/03/22/guia-obs.html.

las transmisiones de live coding en línea y, a su vez, fomentó el contacto entre quienes participaron.

El evento de transmisión para celebrar el Aniversario 16 de Toplap fue nombrado *Eulerroom Equinox 2020* y se realizó alrededor de la fecha del equinoccio de primavera de ese año entre los días 19 y 22 de marzo. Para tal ocasión utilicé de manera distinta la clase SonoTexto: en lugar de grabar fragmentos del sonido ambiente, ingresé la señal de una tornamesa y una caja de ritmos a través de mi tarjeta de sonido junto al sonido de algunos instrumentos de percusión capturados con un micrófono. Asimismo, con una cámara web inserté en un recuadro de la transmisión el espacio de mi departamento desde donde toqué. Transmitir por red requiere recursos distintos a los utilizados cuando se toca en un escenario de forma presencial; es necesario contar con una señal de internet estable con buen ancho de banda, una computadora o dispositivo digital para seguir el evento, así como la computadora para tocar y realizar la transmisión. Otro cambio sustancial en esta situación es el desplazamiento del escenario a la pantalla.

La descripción de este evento denota una aproximación diferente a los realizados en la etapa presencial. Por ello, se hace necesario presentar algunas consideraciones que lo relacionan con el planteamiento de la tesis: 1) el uso de la clase se desplaza de lo presencial a lo virtual, lo que pone a prueba su diseño en una situación creativa distinta, 2) la transmisión en línea construye el espacio virtual a través del encuadre, 3) hay una contracción del espacio sonoro pues el micrófono se emplaza cercano a los instrumentos (tercer tipo de captura explicado en la *Figura 4.1*), 4) se capturan señales directas, como la tornamesa, lo que minimiza la presencia acústica del lugar y 5) la dinámica de grabación en el momento persiste. Esto muestra que la transformación de la práctica artística no solo depende de la interacción con el desarrollo tecnológico, también de las circunstancias.

El siguiente ejemplo ocurre en mayo de 2020. Durante el confinamiento, el colectivo de live coding Toplap Barcelona inició una serie de transmisiones por su canal de YouTube llamada *8:08pm La hora del livecoder*[66]. El título hace alusión al horario en que terminaba el

66 8:08pm La hora del livecoder II https://youtube.com/playlist?list=PLDvUWb-gizIJz3JQ7YOpeIJxCKvbTtDXY.

reconocimiento público de la gente en confinamiento hacia los trabajadores de la salud en diversas ciudades de Europa quienes recibían aplausos cuando regresaban a sus casas después del trabajo[67]. En Barcelona, este acontecimiento sucedía a las a las 8:00 p.m. y ocho minutos después comenzaba la transmisión. Mi participación en esa serie fue el 6 de mayo de 2020, ocasión en la que utilicé SonoTexto al final de la transmisión (véase alrededor del minuto 46:10)[68] para capturar sonidos de la avenida Eje Central que colinda con mi departamento en la Ciudad de México.

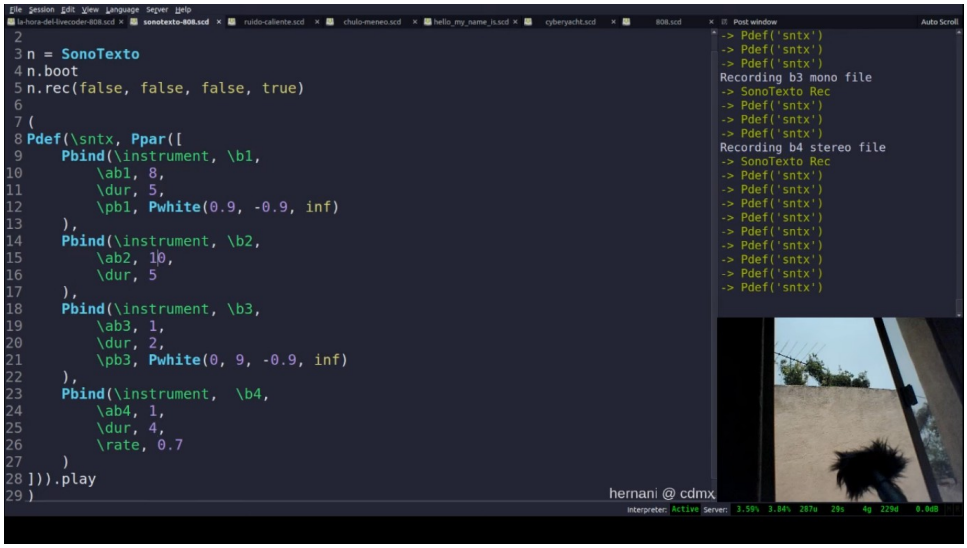



*Figura 4.21. 8:08pm La hora del livecoder. Presentación en línea, 6 de mayo de 2020. Captura de pantalla del canal de Toplap Barcelona.*

El siguiente ejemplo ocurre el 3 de julio de 2020 dentro de la serie de conciertos de música electrónica experimental en línea *Campamento Extendido: <impendingVoid/>* organizado por el sello chileno Posternura Records. Esa presentación la realicé con los sonidos guardados en la carpeta <sonotexto> en lugar de grabarlos durante la transmisión. Aquella ocasión, no fue una presentación partiendo de cero, como en los ejemplos anteriores, sino una con código preparado y con sonidos grabados en ocasiones anteriores. Tampoco utilicé patrones

67 https://es.wikipedia.org/wiki/Aplauso_por_los_trabajadores_de_la_salud#Espa%C3%B1a.
68 https://youtu.be/kFyNxaVP0Yo.

para organizar el sonido, sino que trabajé modificando los parámetros de los buffers generados por un diccionario que cargan los sonidos guardados en la carpeta <sonotexto> cuando es llamado el método `read`. De esta manera, la preparación del concierto consistió en escuchar y seleccionar los sonidos que coloqué en varias tareas o `Tdef` encargadas de leer en bucle los buffers de sonidos cargados en el diccionario. Durante la presentación modifiqué el código de los `Tdef` previamente escritos para tocar, lo que me permitía modelar una secuencia de cambios sobre los parámetros de la clase `PlayBuf`. A partir de ese concierto pensé crear una clase para la lectura de los sonidos almacenados que me permitiera continuar desarrollando la clase SonoTexto en diferentes módulos. Con tal objetivo comencé a diseñar la clase SampleTexto.

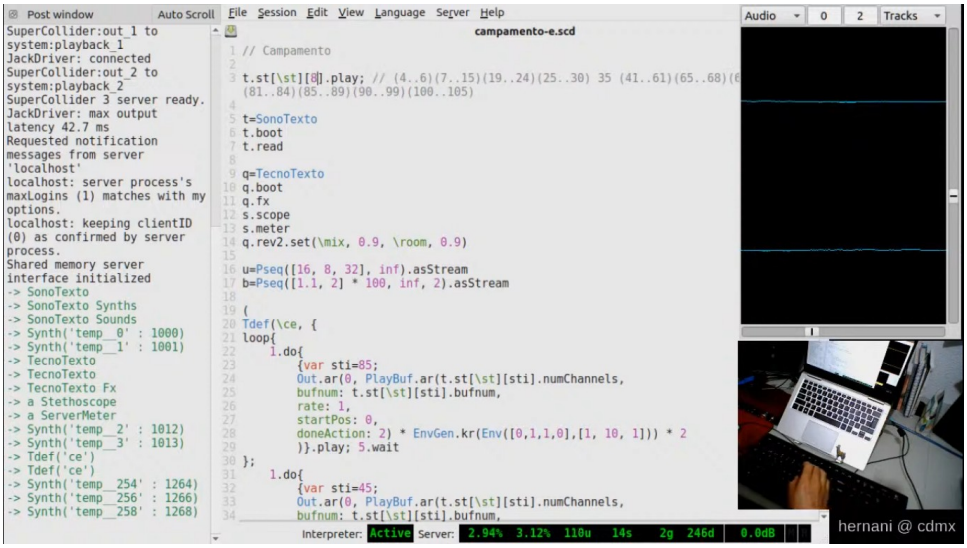


*Figura 4.22. Campamento extendido: <impendingVoid/>. Captura de pantalla del canal posternura records.*

La transmisión en línea para *Campamento Extendido* se apoyó en la función de guardar sonidos en el disco duro con el método de escritura `write` de la clase SonoTexto. La tarea de este método cambia la relación con los espacios y las situaciones del concierto pues transfiere la temporalidad de la grabación y reproducción de sonido durante la presentación en vivo a una en la que los sonidos grabados son reproducidos de manera posterior. El método `write` cambia la idea de comenzar desde cero, desvincula el sonido del lugar y ofrece la posibilidad de preparar

código para tocar, lo que desplaza la intención de improvisar a la de componer. De esta manera, el código se prepara, se buscan soluciones y se corrigen errores antes del concierto. Esta aproximación marcó las siguientes presentaciones en las que me incliné por preparar el código para tocar en vivo. Tal decisión marca la transposición entre la programación exploratoria y prescriptiva en mi práctica artística; si bien al crear música con código preparado no implica el desarrollo de un objeto computacional, aquí ya busco estructurar los eventos. Esto muestra que la programación de la presentación del live coding también puede ser prescriptiva.

El último ejemplo que presento es la transmisión en línea para el evento *Algorave Brasil 2020* el 13 de diciembre de 2020[69]. El evento fue organizado por el colectivo brasileño con el mismo nombre. Para la presentación utilicé la clase SampleTexto con la intención de acceder a los sonidos guardados en la carpeta <sonotexto>.

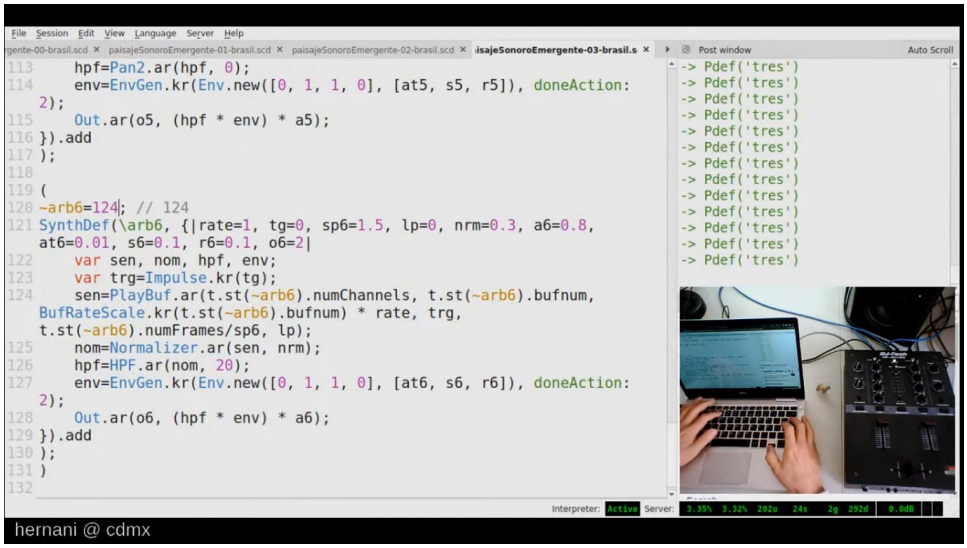



*Figura 4.23. Captura de pantalla del canal Algorave Brasil. 13 de diciembre de 2020.*

Esa ocasión me enfoqué en una mayor preparación del código e incorporé la manipulación del sonido desde una mezcladora de DJ. Para ello, trabajé con dos archivos de código preparado con los que interactué como si fueran dos tornamesas. En esa presentación la

69 Transmisión para Algorave Brasil 2020 por el canal del colectivo https://youtu.be/tMRAJQ0LWAA?t=12129.

práctica de live coding se acercó más al diseño de una interfaz, trajo elementos de prácticas como el *scratch* del tornamesismo, hubo una intención de dividir el código en piezas musicales y partió de un archivo de código utilizado anteriormente en el concierto del XV Coloquio de Alumnos del Posgrado en Música.

Como se escucha en los videos, los ejemplos de esta etapa presentan una forma estructurada de manera rítmica con sonidos grabados en diferentes momentos. La posibilidad de acotar cada sonido con envolventes temporales permitió el enfoque en fuentes sonoras particulares como instrumentos musicales, golpes sobre superficies, voces o el entorno personal. De esta manera cada sonido fue organizado rítmicamente con variaciones de altura y duración de sus envolventes. A diferencia de las masas sonoras de la etapa presencial, las presentaciones en línea tienen un carácter incisivo y rítmico.

Por otro lado, la necesidad de transmitir eventos por plataformas como YouTube y de realizar conferencias y encuentros por sistemas de videoconferencia como Zoom y Jitsi posibilitó que eventos académicos y encuentros artísticos se realizaran abiertos a todo el mundo, lo que permitió una participación fuera de la presencia geográfica. El año 2022, en el que culminó este trabajo, inició con un posible control de la crisis sanitaria con el avance sustancial de la aplicación de vacunas, el relajamiento de las medidas de algunos países, el endurecimiento en otros, rebrotes inesperados y reglas heterogéneas en los protocolos de viaje, movilidad, cuidado e ingreso a cada país. Bajo este panorama, y en el momento que escribo este párrafo (agosto de 2022), algunos países permiten organizar conciertos presenciales con pequeño aforo o al aire libre y otros han vuelto a los conciertos de gran formato bajo medidas y protocolos sanitarios. No obstante, el paulatino regreso a este tipo de actividades, varios festivales, congresos y encuentros organizados anteriormente de manera presencial continúan emitiendo convocatorias para realizarse en línea o en modalidades híbridas.

Es el caso de los conciertos de live coding en línea que se realizaban ya bajo esta modalidad antes del confinamiento, y que a partir de este suceso se volvieron más comunes. Estas transmisiones, generalmente se transmiten en vivo por canales de YouTube o Twitch. Para ello, cada live coder prepara su pantalla de transmisión y decide qué elementos mostrar. La convención es transmitir el archivo donde se escribe el código fuente junto al sonido y la imagen

que este genera. A veces se agrega un recuadro que muestra la cara, las manos o el cuerpo del live coder mientras toca en su espacio de trabajo. El sonido y la imagen se transmiten desde la computadora personal a la plataforma de transmisión de video y desde ahí se ven y escuchan en las computadoras y dispositivos de la audiencia conectada a la plataforma en el momento de la transmisión. Una particularidad es que quienes ven y escuchan el concierto en línea también pueden escribir mensajes en el chat de la plataforma en ese momento. En este sentido, las computadoras interconectadas forman el espacio del concierto al que se accede desde diferentes geografías coordinadas por el estándar de tiempo universal UTC o *Coordinated Universal Time*[70]. Las transmisiones quedan almacenadas en los canales de YouTube, Twitch o Archive para quien desee verlas posteriormente y, en esos casos, quitando el factor en vivo, se reproducen de igual forma en la que fueron realizadas. Es decir, no es un video de registro de una presentación en un espacio físico lo que se transmite sino la adaptación del espacio digital en la que cada receptor tiene una experiencia distinta dependiendo del hardware con el que recibe la transmisión y del espacio donde se encuentra emplazado.

En el tránsito de nueva cuenta hacia las actividades presenciales surge la pregunta sobre si los cambios en las formas de hacer y presentar música durante el confinamiento traerán un nuevo planteamiento al escenario presencial. En respuesta a lo anterior, creo que las actividades que comenzaron a realizarse en línea durante este periodo permanecerán en alguna medida, junto a conciertos, reuniones, seminarios y encuentros académicos presenciales y en los llamados “eventos híbridos”. En el caso específico del live coding, la transmisión de presentaciones en línea continuará como sucedía desde antes de los confinamientos. Lo que se suma es la posibilidad de realizar encuentros, cursos y conferencias a la distancia pues esto ha mostrado el alcance de conexión entre personas y comunidades.

70 El sitio web timeanddate explica que UTC es un estándar para el tiempo civil con base en un reloj atómico de alta precisión combinado con la rotación de la Tierra. Véase https://www.timeanddate.com/time/aboututc.html.

## 4.4 Fragmentos de sonido para la improvisación

Durante los eventos presenciales, la práctica artística de SonoTexto se enfocó en la grabación de *fragmentos de sonido* del entorno acústico de espacios específicos. Tales fragmentos, al igual que la presencia del individuo en el lugar, desaparecen de la memoria temporal de la computadora. Los sonidos referidos ocurren en entornos con su propia identidad acústica, la cual activa la escucha del live coder para grabarlos en esos espacios al teclear el mensaje de grabación `rec` de SonoTexto. La exploración de la grabación con esta técnica se puede extender fuera del espacio. Es decir, la grabación se puede enfocar en fuentes sonoras como instrumentos musicales o el paisaje sonoro. Por otro lado, la intención de grabar fragmentos de sonido que desaparecen de la memoria sufre un cambio de perspectiva al guardarlos en el disco duro. Esto último crea un archivo de sonidos que puede utilizarse para realizar live coding de manera posterior a la acción de grabar.

En los casos anteriores, la duración de los fragmentos grabados con SonoTexto, 5 y 10 segundos, responde a la decisión técnica y cognitiva de manejar el cúmulo de tiempo que ello implica en una situación en vivo y a trabajar con archivos de sonidos cortos de manera posterior. Así, la duración de los sonidos es manejable por la memoria temporal de la computadora durante cada presentación, fácil de explorar en el tiempo que dura esta y accesible para escuchar los sonidos archivados. Si bien las duraciones de los sonidos fueron decididas con base en la eficiencia de cómputo durante una presentación en vivo, se puede argumentar que estas duraciones son manejables por computadoras actuales de escritorio, portátiles y de placa.

Pero lo anterior no solo atañe a la capacidad de la memoria de una computadora, también a la capacidad de retención de la memoria humana. Al respecto, Mineault (2021) menciona que en el campo de las neurociencias la memoria se distingue en memoria de trabajo y memoria a largo plazo. Este neurocientífico computacional explica que el código fuente es leído por un área del cerebro humano de múltiple demanda relacionada con matemáticas, lógica y resolución de problemas. Esta área, dice, tiene relación con la memoria de trabajo que retiene información por periodos cortos de tiempo. Sugiere que al programar lidiamos con mucha información relacionada a variables, funciones, bucles computacionales y más. Por ello, el autor recomienda

trabajar con convenciones de la escritura de código que liberan a nuestra memoria de trabajo de recordar muchos detalles. En la programación en vivo el cúmulo anterior no llega a suceder del todo pues el live coder utiliza sus propias convenciones para escribir código fuente que a veces cambian de presentación en presentación. Lo anterior, sin embargo, se acumula junto a la información sonora que debemos retener temporalmente para improvisar con sonido grabado. Ya sea por una razón técnica o por una cognitiva, la decisión de utilizar fragmentos cortos de sonido ha determinado la forma de improvisar con SonoTexto.

Lo anterior está vinculado a la improvisación, una práctica de creación musical que por sí misma abarca múltiples formas. El compositor e improvisador, Wade Matthews (2012), describe en particular la llamada improvisación libre como una “sin previo plan, estructura o acuerdo” (p. 19). Matthews reconoce que hablar de improvisación depende de un contexto o tradición improvisatoria. En este sentido, para el autor, la persona que improvisa crea música bajo el conjunto de reglas de la tradición a la que pertenece. Estas reglas son de carácter oral, pues como menciona Matthews, la improvisación: no se escribe, no se fija, desaparece en el momento de tocar, no admite cambios y correcciones, conlleva inmediatez, conversación, tiene una función social, es un proceso, asume riesgos y explora el momento.

Si bien todas estas características podrían definir al live coding, Matthews apunta a una en particular que contrasta con la naturaleza del live coding: la forma no escrita de la improvisación. Este contraste deriva de que el live coding se realiza precisamente con la escritura de código fuente. Julian Rohrhuber y Alberto de Campo (2009), dos artistas y académicos que han aportado en gran medida a la reflexión del live coding como práctica improvisatoria, están conscientes de esta particularidad. Los autores argumentan, en el artículo *Improvising Formalisation – Conversational Programming and Live Coding* (2009), que la programación puede ser una práctica improvisatoria. Lo anterior, dicen, si consideramos el cambio de la relación entre *formalización* y *proceso* que surge cuando vemos la programación como un método de reflexión; esto es, un procedimiento de reinterpretación de reglas en el que la formalización es improvisatoria. Específicamente, los autores se refieren a la actividad de reprogramar un programa de música mientras corre. Sin embargo, apuntan que, para que sea esto posible se necesitan extensiones en los lenguajes de programación. De esta manera, los autores

consideran que la computadora llegó a ser un instrumento de improvisación en el momento que fue posible cambiar e intervenir parámetros en tiempo de ejecución. A partir de ello, es que Rohrhuber y De Campo (2019) llegan a la idea de que la improvisación en el live coding es la descripción formal de un proceso.

Por su parte, Carolina Di Próspero (2015) describe al live coding como una actividad de improvisación, puesto que es de carácter inacabado, se enfoca en el proceso creativo y tiene una intención colectiva/relacional. Esta última puede ocurrir incluso de manera extramusical en el intercambio de conocimientos que se da en las comunidades de práctica. Algo crucial, que señala la autora, es que para improvisar con un lenguaje de programación el tiempo de reacción/intervención dentro del loop de retroalimentación de la escritura de código debe ser mínimo. Lo anterior hace referencia a la posibilidad de implementar los cambios realizados al código de manera inmediata para dar continuidad al sonido.

Como se observa, la noción de improvisación comparte varias características con el live coding de las que resalta el énfasis que hay en el proceso y la conversación. Asimismo, los aspectos tecnológicos apuntan a la introducción de mecanismos que permitan el continuo sonoro, lo que lleva a un proceso de escritura y pensamiento del código en ciclos. Como mencioné antes, este proceso pone en juego la memoria para poder conversar, tomando la idea de Rohrhuber y De Campo, entre la escritura de código y el sonido producido. La capacidad de lidiar con código y escuchar su resultado se ve cuantificado más que en la cantidad de código escrito en el número de estructuras o bloques que contienen los parámetros a modificar.

Respecto a lo anterior, he observado que es común trabajar con 4 bloques de código, de información visual o sonora, ya sea en una presentación de live coding o en el diseño de objetos computacionales para este fin. Por ejemplo, la función `render()` de Hydra permite trabajar con 4 salidas de video; por otro lado, algunas presentaciones de live coding muestran cómo sus practicantes llegan a un punto de saturación con 4 bloques de código en tiempos de 10 y 20 minutos. De lo anterior puedo mencionar la presentación en línea de Yaxu + Hellofoodcat en el evento Cyber Yacht[71], el performance poético de Jessica Rodríguez en la transmisión en línea

71 Yaxu + Hellofoodcat: https://youtu.be/-QQ3W3rdEI8?t=1932.

New Moon[72], así como las presentaciones en vivo *Weathery* de Gerard Roma[73] y *Live coding with integrated visualization of code information and sound impression* de Hiroki Matsui en el ICLC 2020[74]. En todas ellas se observa el manejo de cuatro bloques de código que sostienen o desarrollan la presentación. Esto mismo se puede observar en el diseño de SonoTexto que trabaja con 4 buffers de sonido y en las presentaciones que realicé de manera presencial en las que el punto de saturación ocurre con 4 bloques de código insertos en un Pbinds[75].

Pero, ¿por qué trabajar con fragmentos de sonido ambiente acotados a la duración del buffer en lugar de hacerlo con grabaciones de campo o de estudio que pueden ser cortadas de manera precisa con un editor de sonido? La decisión de hacerlo de esta manera responde a las posibilidades técnicas que el objeto computacional ofrece a la práctica artística; en este caso, el corte de sonido es delegado a un momento de escucha en el que decidimos cuándo iniciar una grabación y cuyo final está determinado por la duración del buffer. Quizá la metáfora es la de una cámara fotográfica que captura diferentes momentos de un mismo espacio a diferencia de una cámara de video que permite un captura de manera continua. En la primera situación decidimos el momento de obturación y la cámara finaliza el proceso, en el segundo decidimos cuándo comenzar a grabar y cuándo terminar. SonoTexto estaría relacionado al primer caso.

Lo anterior muestra cómo una decisión técnica transforma la práctica artística en una acción que involucra la escucha del humano y el cómputo relacionado para llevar a cabo esa acción. Es decir, la implementación de buffers con duraciones específicas delega al live coder el momento de iniciar la grabación, a través de su escucha, y a la computadora el momento de terminarla con base en el cálculo programado. Es con esto que me refiero en mi planteamiento al desarrollo de objetos computacionales que actúan y se transforman junto a la práctica artística, aunque en este caso se revela que esto también ocurre al revés.

El trabajo con fragmentos sonoros posibilita grabar, reproducir hacia adelante o en reversa, leer diferentes puntos del buffer, modificar el envolvente sonoro, la velocidad de reproducción (que repercute en la altura) y la posibilidad de reproducir varias copias del mismo

72 Jessica Rodríguez: https://youtu.be/Jb69NcKtqjk.
73 Gerard Roma: https://youtu.be/hH0KZ2rSLRA?t=2800.
74 Hiroki Matsui: https://youtu.be/hH0KZ2rSLRA?t=3543.
75 Hernani Villaseñor: https://youtu.be/rejN3CHMbpw?t=2306.

sonido. Dichas posibilidades son heredadas de los objetos de SuperCollider que se encuentran encapsulados dentro de las funciones de la clase. De esta manera, un sonido de 5 segundos puede ser reproducido en toda su extensión tal como fue grabado o en diminutas porciones de diferentes duraciones, alturas y puntos de lectura. Tales posibilidades pueden ser catalogadas desde lo que María Herrera Lima comenta acerca del entendimiento general de la estética como una "perspectiva que establece criterios de clasificación y valoración (formales/expresivos)" (2019, p. 285). En el caso de las posibilidades de manipulación de los fragmentos sonoros mencionados arriba, según lo que comenta Herrera Lima, pueden verse como criterios estéticos. Tales criterios vienen desde la música concreta de los años 1950 cuando se introdujeron las posibilidades técnicas de grabación, reproducción y manipulación de sonido en cinta.

Por su parte, SonoTexto se involucra con la creación musical con sonidos grabados a partir de dos aspectos: la grabación programática de sonido en el momento y la transformación algorítmica de estas grabaciones con la escritura de código fuente. En este sentido, el sonido grabado es colocado en un bucle no lineal controlado por algoritmos presentes en los patrones de los lenguajes de programación musicales. Al respecto, Nierhaus (2009) menciona que algunos algoritmos utilizados en la música, a diferencia de los algoritmos que definen la solución de un problema en un número determinado de pasos, son indeterminados, introducen probabilidad, procesos estocásticos y a veces no terminan. Esto hace que el sonido controlado por procesos algorítmicos musicales, como los descritos por Nierhaus, arroje resultados impredecibles en la secuencia temporal en la que son organizados. Este tipo de aproximación se observa, además de SonoTexto, en presentaciones que combinan el live coding con la grabación de campo como los casos de Lucy Cheesman[76] y Niklas Rappel[77]. Ambos live coders, a raíz de la pandemia, han explorado el live coding en lugares naturales abiertos fuera del escenario o el estudio. Ambos utilizan código para grabar sonido en el momento de manera programática para incluirlo en sus presentaciones. Cheesman utiliza un *looper* programado con TidalCycles y Reppel ha

76 *Live from Ecclesall Woods* de Lucy Cheesman (heavy lifting) (2020) es una presentación en la que emplea la grabación de sonido en el momento para hacer live coding con TidalCycles: https://youtu.be/yqOR7ULUQEE.

77 Bodies of Water – Fieldcoding Exercises for Code de Niklas Reppel (2021) es una presentación de live coding en la que Reppel graba sonido en el momento con el lenguaje de programación de su autoría Mégra: https://youtu.be/SpzrDyASXTg?t=1871.

desarrollado un lenguaje de programación llamado Mégra al que incluyó funciones para grabar en el momento.

Por su parte, SonoTexto aporta una reflexión basada en la práctica artística y tecnológica, sobre la relación de la grabación de sonidos en el espacio y el diseño del objeto computacional para ello. Es decir, el objeto computacional ha sido diseñado a partir de la idea de moldear el sonido de un lugar durante una presentación de live coding y no como un objeto que permite grabar sonido independientemente de la situación. Esto responde directamente al planteamiento inicial de desarrollar objetos computacionales vinculados a la práctica artística.

Las aproximaciones al sonido descritas anteriormente ocurren de manera distinta al trabajo con cinta de la composición electroacústica, acusmática o concreta que Beatriz Ferreyra describe en la entrevista para ANTIDIÁSPORA (2015)[78]. La compositora de música electroacústica habla sobre el sonido y cuenta su aproximación técnica y estética. Ferreyra deja entrever la disposición lineal de múltiples sonidos organizados en una sesión de trabajo de una de sus piezas. El programa DAW que utiliza, como analogía de la cinta, muestra parte de las decisiones compositivas, y a veces improvisadas, de las que ella tiene control en un formato en el que puede colocar los sonidos de forma manual.

En contraste, la forma de manipular el sonido con código fuente escrito en el momento sucede al escribir comandos que refieren las posibilidades estéticas mencionadas anteriormente y al indicar el valor numérico con el que se realiza cada modificación. Así, por ejemplo, pensamos un sonido en términos de los valores de su duración, velocidad de reproducción y repetición. Lo anterior ocurre en un proceso de escritura de código que toma tiempo pensarlo y teclearlo; así que un recurso para mantener el continuo del sonido en una presentación de live coding es ponerlo en el bucle de un patrón o rutina mientras se escribe cada modificación. La comparación del live coding con la música electroacústica no es ociosa. Parece que con la llegada del live coding musical se abrió un campo nuevo al trabajo con sonido que permite manipular muestras de audio y grabaciones de campo de manera algorítmica. Sin embargo, responde a procedimientos parecidos a los realizados en la música concreta aunque en medios diferentes, con resultados distintos y en sus propios circuitos de producción y difusión.

78 Documental ANTIDIÁSPORA|Beatriz Ferreyra: https://youtu.be/HW3olqNOsBA.

A partir de lo anterior, la programación de música no se dedica exclusivamente a la síntesis de sonido basada en la notación, esta comparte ideas con la teoría de la música electroacústica centrada en el trabajo con sonido, o señal. En este sentido, los lenguajes de programación, insertos en el hardware de la computadora, pueden controlar las entradas y salidas de sonido como son los micrófonos y las bocinas incorporadas. Esto crea una situación particular pues la música electroacústica, acostumbrada a la linealidad libre de notación en la que el material sonoro es su propio medio de inscripción, va a lidiar con el sistema de símbolos del código fuente y con la implementación de las ideas musicales en algoritmos que controlan la organización del sonido en el tiempo. Así, el material sonoro de la música electroacústica se somete al control del algoritmo en lugar de la linealidad de la estación de trabajo DAW. En esta situación se contrasta la escucha que se da en la música electroacústica y en el live coding, en la primera ocurre de manera previa desde la preparación de los materiales sonoros. En cambio, en el live coding la escucha ocurre después, primero programamos y luego escuchamos, esta situación se ve plasmada en el discurso de la incertidumbre y riesgo al programar del live coding.

En este sentido, en el momento de programar, el live coder se relaciona con el sonido a través de la mediación textual del código fuente y la escucha del resultado de sus acciones programáticas. La destreza de escritura y la memorización de los comandos y sonidos lo hacen posible. Lo interesante del live coding es que se parece a un sistema de composición por notación debido a su relación simbólica con el sonido, pero la dinámica es la de la improvisación libre. A su vez, en el momento de trabajar con sonido grabado, el live coding se acerca a la estética de la música electroacústica y de otras prácticas que trabajan con sonido.

Por otro lado, además de las posibilidades técnicas referidas como criterios de valoración estética, trabajar con sonido distinto al que viene incluido con los programas posibilita una forma de franquear la barrera de la expresión personal y entrar al software desde la inclusión de sonidos propios. Por ejemplo, Rafrobeat, Jessica Rodríguez o Khoparzi utilizan el lenguaje de patrones TidalCycles con sonidos tomados de sus contextos. Ese punto de apertura, en TidalCycles y otros lenguajes de programación, permite ingresar sonido para transformarlo con sus estructuras computacionales y algorítmicas. En este sentido, las narrativas musicales y sonoras se realizan a partir del sonido situado, como datos, y de la manipulación algorítmica. Esto implica ingresar

contextos sonoros personales, paisajes sonoros, instrumentos musicales, estilos de música y oralidad al sistema algorítmico propuesto por cada lenguaje de programación. Pero ¿qué pasaría si, además de datos sonoros, ingresamos objetos computacionales? ¿se escucharía? Esto es algo que cuestionan Collins et al. (2003), quienes reconocen la idiosincrasia implícita en el software pero aseguran que es difícil detectarla en un nivel sonoro.

Contrario a esto, Marino (2020) cuenta que el artista Jon M. R. Corbett, desde un ámbito visual, desarrolló un programa con el lenguaje de programación Processing que le permitió simular el bordado de retratos basado en el hilado de cuentas de la cultura Métis a la que el artista pertenece[79]. Marino (2020) se refiere a este proyecto como una aproximación a la codificación de la epistemología Métis, presente en la lengua Cree propia de esta cultura y expresada a través del hilado de cuentas, que es trasladada a los procesos algorítmicos de un lenguaje de programación. Este ejemplo da cuenta, en el ámbito visual, de la exploración de la implementación de modos de expresión a la lógica de los lenguajes de programación. Aquí la aproximación no se da en el desarrollo de un nuevo lenguaje de programación sino en la abstracción de una forma cultural.

El ejemplo anterior permite observar la lógica de programación expresada en un resultado visual, lo que quizá es más difícil distinguir en un caso sonoro. En cuanto a SonoTexto, ciertamente resulta difícil escuchar a través del diseño de los objeto computacionales, aquí más bien el objeto computacional se manifiesta a través de los sonidos grabados del entorno sonoro donde son ejecutadas sus clases.

Con el fin de mostrar lo anterior, la *Figura 4.24* muestra un fragmento de código fuente que permite explorar el live coding con SonoTexto. Para ello, se instalan las clases y, posteriormente, el código del ejemplo se puede copiar en el IDE de SuperCollider. A manera de algoritmo verbal se siguen estas instrucciones: 1) abre SuperCollider y copia el código del ejemplo, 2) declara la primera línea de código y espera a que inicie el programa, 3) declara la siguiente línea para crear el objeto SonoTexto en la variable `t`, 3) declara la línea `t.boot`, 4) espera a escuchar un sonido interesante y declara la línea `rec`, también puedes emitir sonidos con

79 Según *The Canadian Encyclopedia* Métis se refiere a un grupo étnico, conformado por el mestizaje de indígenas y europeos, que vive en Canadá. Cree es uno de los grupos nativos que formó parte del mestizaje y es la lengua que hablan sus descendientes. https://www.thecanadianencyclopedia.ca/en/article/metis.

la voz o con algún objeto cercano, 5) a continuación declara el `Pdef` y escucha la acción algorítmica del código sobre el sonido que acabas de grabar, 6) modifica algunos valores del código y vuelve a presionar la línea de grabación `rec`, 7) escucha el resultado, 8) por último cambia la palabra `play` por `stop` y declara el `Pdef` para terminar la pieza.

```
s.boot;
t=SonoTexto;
t.boot;
t.rec(1);

(
Pdef(\sono,
   Pbind(\instrument, \b1,
      \dur, Ptexto([Pseries(0.125, 0.125, inf)], 1.5, 0.125, inf),
      \rate, Pseq([1, 1.1, 0.75].mirror, inf),
      \sb1, Pseq([0.2, 0.5, 0.1], inf),
      \rb1, Pseq([0.5, 0.2, 0.3], inf),
      \pb1, Pwhite(-1.0, 1.0, inf),
      \st, ~buf1.numFrames * Ptexto([1.1, 1.5], 1, 0, inf),
)).play
)
```

*Figura 4.24. Pieza algorítmica con SonoTexto. Graba sonido en el buffer uno, lo reproduce con Pdef y puede ser modificado.*

Al realizar esta pieza, estoy proponiendo una forma algorítmica junto a los objetos computacionales que la soportan. Esta pieza "sonaría" de manera similar las veces que se ejecute, lo que cumple con la condición determinista algorítmica que menciona Nierhaus (2009): a un número de entradas similares corresponden las mismas salidas. Pero no va a sonar exactamente igual porque las entradas van a ser diferentes en cada caso, pues la pieza corre en diversas computadoras con sus respectivos micrófonos y en contextos sonoros distintos. Es decir, las entradas corresponden al sonido situado de cada persona y a la escucha que le permite decidir cuándo accionar el comando `rec`.

Las etapas de desarrollo tecnológico y práctica artística por las que ha pasado el proyecto SonoTexo durante la investigación plantean posibilidades técnicas e intenciones de creación musical. Asimismo, están enmarcadas por el contexto cultural y por las circunstancias derivadas de la pandemia de COVID-19. Al inicio de la investigación, la clase SonoTexto posibilitó grabar sonido en el momento de cada presentación presencial, una grabación de carácter efímero ligada a la improvisación libre. Posteriormente, la clase SampleTexto permitió organizar, escuchar, explorar y seleccionar los sonidos que fueron guardados en alguna presentación presencial o aquellos grabados de forma diferida. Este cúmulo de sonidos, a diferencia de aquellos que desaparecen al terminar la presentación, planteó una intención de improvisación estructurada y composición que se intensificó durante el confinamiento. Finalmente, la clase Ptexto me permitió organizar el sonido en el tiempo de la creación musical bajo el esquema de un patrón. De esta manera, las clases de SonoTexto comienzan con la intención de ser utilizadas durante una presentación en vivo y poco a poco son utilizadas como objetos de grabación, catalogación y organización de sonido en momentos diferidos a la presentación presencial. Lo anterior describe un orden cronológico en el que el desarrollo de los objetos computacionales está ligado a cambios de intención de la práctica artística.

Así, la práctica artística de SonoTexto se conecta con la estética de la música computacional que trabaja con sonido grabado donde el resultado musical tiene que ver con las posibilidades técnicas de grabación del objeto computacional abierto. Esto produce música algorítmica improvisada con sonido grabado en el, y del, lugar. Además, cada intención va dictando las ideas creativas junto a las posibilidades técnicas, estéticas y circunstanciales. Esto muestra que, el cambio de perspectiva del proyecto no solo obedece a cuestiones tecnológicas y artísticas, también es consecuencia de situaciones como la aparición de nuevas enfermedades y de las políticas sanitarias que cada país adopta al respecto. Todo esto impactó al desarrollo tecnológico de los objetos computacionales y a la consecuente transformación de mi práctica artística durante el proceso de la investigación.

# 5 SonoTexto: investigación orientada a la transposición entre práctica artística y desarrollo tecnológico

El arte y las humanidades abogan por incluir la programación exploratoria y prescriptiva en sus campos. A su vez, las ciencias de la computación interceden para entrar a las artes digitales y realizar una práctica artística basada en el desarrollo tecnológico. Términos como humanidades digitales (Hayles, 2012), artista-programador (McLean, 2011) y programación exploratoria (Di Próspero, 2015, 2017 y 2019) concilian tales aproximaciones.

El interés por la programación y el código de computadora que tienen campos distintos a las ciencias de la computación, como el arte computacional y la investigación artística en tecnología musical, dejan ver un proceso de transformación de las prácticas artísticas y de las investigativas. Algo que el campo de la tecnología musical asienta como posibilidad aunque no existan modelos y metodologías concretas para este fin. Al igual que la investigación artística, la tecnología musical construye la producción de conocimiento con base en metodologías establecidas en las ciencias, las humanidades y las ciencias sociales.

Este trabajo fue realizado bajo la adscripción al campo de la tecnología musical y a un conglomerado más amplio que incluye diferentes aproximaciones de investigación musical propias de las humanidades. En este sentido, el giro computacional y epistémico, que involucra al encuentro de las humanidades con las ciencias de la computación y a las prácticas artísticas como productoras de conocimiento dentro de una universidad, sirvió como marco para llevar a acabo esta investigación. A su vez, este doble giro ha sido planteado como un movimiento de transposición que hace referencias cruzadas entre las diferentes partes involucradas y no simplemente como un desplazamiento de un punto a otro.

Las relaciones que surgen dentro de la investigación se dieron de la siguiente manera: los modos de hacer de la práctica artística se involucraron con las metodologías para desarrollar software a través de la problemática planteada en este trabajo bajo una perspectiva de investigación artística en tecnología musical. Tales desplazamientos no fueron obvios y tampoco

ocurrieron automáticamente con la apertura del código fuente. Para ello requerí indagar en metodologías y estructuras estandarizadas de escritura de código fuente del ámbito del desarrollo de software y las ciencias de la computación. Sin embargo, el acceso a este conocimiento no fue con la intención de prescribir la creación artística. Más bien, la práctica artística se transpuso con el desarrollo tecnológico para construir conocimiento desde la relación entre la programación exploratoria y la programación prescriptiva. Este capítulo, de carácter conclusivo, retoma las ideas teóricas de la tesis para discutir las conexiones con la práctica artística y el desarrollo tecnológico desde la pregunta central. Asimismo, muestra las categorías creadas, da cuenta de los alcances de la tesis y da pie a las conclusiones finales.

## 5.1 Apertura, abstracción y transposición en la programación exploratoria y prescriptiva

El desplazamiento de la programación exploratoria a la programación prescriptiva en el live coding musical es un movimiento inicial que traslada las ideas que surgen en la práctica artística a las posibilidades técnicas de un lenguaje de programación para construir objetos computacionales. Este desplazamiento entre formas de programar conlleva aprender nuevas maneras de escribir y organizar el código fuente de un programa caracterizadas por el planteamiento y resolución de problemas, la organización de estructuras formales y el seguimiento de convenciones de escritura. Lo anterior sucede a diferencia de un forma de programación caracterizada por una aproximación exploratoria al uso de los recursos de un lenguaje de programación. Una forma de programar basada en probar diferentes combinaciones de dichos recursos y en asumir los posibles errores que surjan durante una presentación de música creada en vivo con un lenguaje de programación.

Visto así, la transposición ocurre entre la práctica artística y el desarrollo tecnológico de un objeto computacional. En ese desplazamiento de ida y vuelta, las posibilidades de la práctica artística quedan encapsuladas como funciones y datos dentro de este objeto, que en el caso del presente trabajo ocurrió en el elemento clase. Para que la transposición no se convierta en un

movimiento de una sola dirección, hay que probar las posibilidades del objeto computacional en la práctica artística y regresar el resultado de la experiencia al desarrollo tecnológico. Así, se establece un ir y venir entre formas e intenciones de programar que ocurre, y nos coloca, en medio del movimiento de transposición.

Este fenómeno también ocurre en la abstracción computacional que, en el desplazamiento de ida y vuelta, nos conduce primero a la disección del objeto computacional para entender su funcionamiento y después al encapsulamiento de sus elementos para poder llevar a cabo la práctica artística con él. Por un lado, la abstracción destapa el contenido de las cajas negras que nos permite analizar la manufactura de una tecnología informática y, en sentido inverso, encapsula las funciones que determinan nuestra práctica artística como se observa en el caso del elemento clase del paradigma de programación orientada a objetos. En este sentido, la práctica artística expone su código fuente en la proyección dirigida a la audiencia durante una presentación de live coding mientras que la programación prescriptiva dicta cómo empacar lo expuesto. Este fue el caso de SonoTexto, una clase que en el inicio consistía en líneas de código preparado expuestas a la audiencia junto al código escrito en el momento, que posteriormente fueron organizadas y encapsuladas en una clase de SuperCollider. Además, el proceso de aprendizaje de una forma de programación distinta a la empleada en la práctica artística facilitó el entendimiento para el diseño de otras clases que expandieron la idea inicial.

El encapsulamiento de código fuente es un procedimiento de la programación de clases que puede sonar contradictorio a la apertura que demanda la transposición entre conocimientos de programación, pues parecería que esta acción está cerrando el código fuente en lugar de abrirlo. Sin embargo, no es así ya que operar el código fuente expuesto de la programación exploratoria durante una presentación de live coding nos enfrenta a una navegación compleja e ilegible que cierra la lectura de código fuente al entendimiento de la audiencia. Al respecto, Rohrhuber y De Campo (2009) mencionan que cuando improvisamos música con algoritmos en el contexto del live coding es difícil detectar si los cambios del sonido son producidos por este mecanismo o por el cambio que realizamos en su código fuente. Sugieren que la elección de un algoritmo complejo quizá no es la mejor opción para lo que llaman *programación conversacional* pues la modificación de su código fuente en el momento de una presentación

musical causará ambigüedad entre los procesos automatizados y la intervención humana. Esto provocaría poca o nula distinción de la relación entre los cambios hechos al código fuente y el sonido que se desenvuelve para la improvisación del live coder y la apreciación de la audiencia.

Aunque el código fuente que resulta de la programación prescriptiva está empaquetado en una clase, su apertura se manifiesta en el repositorio público. Con ello vamos y venimos entre la abstracción alta de la práctica de live coding y la abstracción baja del objeto computacional, entre el abrir de cajas para reescribir su contenido y el cierre de estas para usarlas en el momento de cada presentación. La transposición se encuentra precisamente en ese abrir y cerrar, subir y bajar, así como en la posibilidad de lo abierto que nos permite realizar y compartir ese proceso. La programación de una abstracción más baja coloca la imaginación de nuestra práctica artística dentro de las funciones del software. En este sentido, la transposición no es simplemente una estrategia técnica para empacar y desempacar código fuente, sino la relación que nos permite realizar un movimiento entre formas de pensamiento y creación.

SonoTexto posibilitó la transposición entre dos formas de pensar la programación orientada a la creación musical en esta investigación. A partir de eso, este trabajo propuso un desarrollo tecnológico simple, con recursos estándares de SuperCollider encapsulados en la estructura de la clase. El entendimiento del elemento clase, bajo la guía de desarrollo de SuperCollider, permitió ingresar por las entradas abiertas del código fuente de este lenguaje de programación. Algo quizá obvio para quien viene de la programación estructurada, pero que desde la perspectiva del live coding artístico no lo es tanto, pues dichas entradas aparecen cerradas si no contamos con el conocimiento técnico y todos los elementos para crear música.

En este sentido, la apertura implica un compromiso por mantener la programación prescriptiva sencilla y legible. Mineault (2021) habla de mantener el estilo del código simple mientras que Joyanes Aguilar (2020) propone que sea dividido y modular. El desarrollo de clases pequeñas y modulares basadas en una tecnología abierta y entendible es un modelo que permite la transposición entre lo que imagino que puede sonar en la presentación de live coding, la tecnología que requiero para ello, así como los conocimientos musicales, experiencia artística, gustos y afiliaciones personales. Esto señala que el sesgo que impone la solución computacional no es solo tecnológico sino también estético. A partir de esto, SonoTexto puede definirse como

un objeto computacional abierto que posibilita la práctica artística del live coding con sonido grabado. La simpleza de su diseño traza el espacio de la transposición entre modos de programar y formas de hacer música.

Así, la práctica artística de la creación musical adquiere una dimensión propia enmarcada por las convenciones del lenguaje de programación. Esta se hace más prescriptiva y toma consciencia de los procesos que hay debajo. Muestra sesgos tecnológicos y estéticos. Adquiere el control para grabar de manera programática. Involucra al objeto computacional como parte de la práctica artística. Por otro lado, el objeto computacional revela la idiosincrasia de su desarrollo al ser expuesto en la práctica artística. Esto muestra una solución personal encapsulada en un pequeño sistema computacional para la creación musical.

## 5.2 De la práctica artística al desarrollo tecnológico

Este trabajo ha planteado que el live coding, aunque es una práctica artística computacional basada en la apertura del conocimiento y el código fuente, establece una barrera entre sus practicantes y quienes desarrollan los lenguajes de programación utilizados que termina por prescribir la creación artística y musical. Ante ello, la transposición entre conocimientos de programación y el acceso a niveles de abstracción es una alternativa para implementar ideas artísticas y modos de hacer en los lenguajes de programación establecidos. Bajo este planteamiento, el estudio y análisis de la investigación arrojó lo que expongo a continuación.

El trabajo empujó la práctica artística del live coding hacia el desarrollo tecnológico con base en un marco teórico de las humanidades digitales, la tecnología musical y las ciencias de la computación. Además, distinguió las categorías *programación exploratoria* y *programación prescriptiva* en el live coding, expuso la barrera que estas generan entre artistas que programan y programadores que desarrollan software y subrayó la intención de la comunidad de live coding por borrarla a través de lo abierto; asimismo, estudió el live coding desde la noción de

abstracción en la computación para analizar la programación exploratoria y prescriptiva en un contexto de apertura tecnológica, artística y científica.

También abordó el desarrollo tecnológico, desde una perspectiva artística, para programar formas de expresión bajo metodologías del desarrollo estructurado del objeto computacional. Lo anterior se realizó de manera específica en el diseño de clases del lenguaje sclang de SuperCollider, lo cual se materializó en las clases SonoTexto, SampleTexto y Ptexto. A partir de ello, señaló, desde la abstracción, cómo un objeto computacional se anida en otro de mayor jerarquía y funcionalidad, y mostró la transposición como una forma de entender el desplazamiento entre las capas del entramado que conforma la práctica artística y el desarrollo tecnológico en el live coding.

Desde lo anterior, programar de forma exploratoria se puede pensar desde un nivel de abstracción más bajo, en el punto donde comienza el desarrollo del objeto computacional junto al proceso de creación de la práctica artística. Es decir, el desarrollo de un objeto computacional es parte de la práctica artística la que, a su vez comienza en el momento de diseñar tal objeto.

Dentro del contexto de SuperCollider, SonoTexto propone capturar el sonido del entorno público y privado de manera programática y organizarlo algorítmicamente con el patrón Ptexto. Esta metodología es prescriptiva no solo por las reglas computacionales, sino también por la forma musical que sigue, una forma ligada a las posibilidades de la música electroacústica. Con ello quiero decir que la prescripción del código fuente estructurado está permeada por modos de hacer de la creación artística que se basan en tradiciones musicales recientes y, a su vez, tienen definidos sus protocolos de creación. Si bien el live coding está ligado a la improvisación, esta no deja de ser una práctica formalizada no solo por las estructuras del código fuente que incorporamos a nuestra práctica artística cuando memorizamos código sino también por las estéticas de las que venimos. Por otro lado, cabe preguntarse si el live coding, para quien se inicia, ofrece una estética propia o se refiere a una técnica de creación musical. En este sentido, la enseñanza de live coding viene de la mano de una forma de entender la creación musical en un contexto específico.

Hablar de una prescripción del código estructurado delega esa característica solo a la programación prescriptiva y supone que la práctica del live coding genera su estética por sí misma. Sin embargo, no es en sí el lenguaje de programación el que nos guía a crear música de cierta manera sino los mecanismos implementados en él que han sido diseñados para resolver problemas estéticos desde una práctica musical particular. El utilizar los recursos de un lenguaje de programación nos llevará por caminos parecidos, incluso bajo el argumento de usar el lenguaje de programación de manera inesperada en la que sin tener conocimientos previos de música y programación llegamos a ciertos resultados. En ese camino el lenguaje de programación nos guía, pasamos por la escritura exploratoria del código fuente, el cual, aunque no necesitamos saber como está hecho, nos demanda escribirlo, declararlo y modificarlo bajo determinadas reglas.

## 5.3 Formas de investigar modos de programar

A lo largo de este trabajo he discutido acerca del conocimiento implícito en los modos de programar en el live coding que posibilitan a quien lo practica expresarse artísticamente y a quien desarrolla los lenguajes de programación diseñar las estructuras que permiten tal expresión. Dicho conocimiento, generalmente de carácter técnico circula en reuniones, foros, tutoriales y repositorios. Por otro lado, esta tesis reflexiona sobre cómo se genera conocimiento desde el live coding bajo la estructura de una investigación académica dentro del campo de conocimiento de la tecnología musical con una perspectiva artística. A partir de lo anterior, esta sección reúne en dos grupos las formas de programación y el tipo de investigación a las que llegué bajo el esquema del presente trabajo.

El primer grupo establece que la programación exploratoria de la práctica artística se encuentra en un nivel de abstracción alto, pues consiste en saber programar para crear arte con un lenguaje de programación. El código fuente de la programación exploratoria en el live coding existe de manera efímera y se desenvuelve temporalmente durante la presentación, es el código que vemos proyectado en la pantalla. El código fuente es el medio de expresión de la práctica

artística, por lo que es una de las partes que conforman al live coding. Implica saber programar con un lenguaje de programación con el objetivo de crear música. Su traducción es interpretada.

Por su parte, la programación prescriptiva del desarrollo tecnológico tiende a estar en un nivel de abstracción más bajo, es decir, encapsula funciones, objetos y datos. Su código fuente queda hospedado en repositorios como GitHub. Es la forma del software antes de ser compilado para operar. Es el objeto computacional que sostiene la práctica artística e implica programar con un lenguaje de programación para desarrollar software.

El código fuente que resulta de la programación exploratoria se transpone con la funcionalidad del código fuente que resulta de la programación prescriptiva. El código exploratorio navega en el código estructurado y encapsulado de clases, funciones, datos y archivos que forman un lenguaje de programación. Aunque el código fuente no es el resultado artístico, este forma parte de la estética de las presentaciones de live coding, es parte del proceso. Es el material textual que conecta las funciones encapsuladas en las clases con la operabilidad de los objetos o instancias de la clase en el momento de realizar live coding. Es precisamente en el momento en que llamamos a la clase para generar su objeto que sucede la conexión del código fuente encapsulado en la clase con el código fuente que será escrito en el acto de la creación musical del live coding.

El lenguaje de programación es un objeto computacional prescriptivo, una pieza de código fuente estructurado que contiene reglas sintácticas, léxicas y semánticas precisas que permiten programar de forma exploratoria en un nivel de abstracción alto. Escribimos y combinamos de muchas maneras las reglas, los comandos y las funciones de la programación para lograr resultados artísticos diversos e inesperados. Sin embargo, aunque el error y el fallo en la escritura son permitidos en el live coding, procuramos utilizar las reglas permisibles del lenguaje empleado. Especulamos con las posibilidades enmarcadas dentro de cada lenguaje de programación para construir formas musicales, visuales, corporales y literarias. A la par, la música y los gráficos son los resultados de la programación exploratoria. El manual, los archivos de ayuda y los tutoriales del programa prescriben la forma de uso. Con este conocimiento podemos explorar diversas combinaciones con las que “improvisamos la formalización” (Rohrhuber y De Campo, 2009) en el live coding.

El segundo grupo establece la categoría de investigación artística en tecnología musical, la cual es posible construir con metodologías de las humanidades y las ciencias de la computación. Este tipo de investigación se define en términos del campo de la tecnología musical y entra a la discusión de la investigación artística. Un terreno enmarcado por las facultades universitarias de arte y música en el que las prácticas artísticas y musicales se juntan con campos académicos y científicos. Y, en este caso, con un campo tecnológico.

Es posible configurar los objetivos de la investigación artística de modo que esta adquiera forma y sentido dentro del campo de la tecnología musical. En otras palabras, la investigación artística es una posibilidad para la tecnología musical. La investigación artística en tecnología musical es una aproximación para aquellos proyectos de investigación musical que reflexionan desde la práctica artística y que en este proceso utilizan, desarrollan y vinculan objetos tecnológicos. Es decir, en la investigación artística en tecnología musical el desarrollo tecnológico no es un medio para llevar a cabo la práctica artística, este forma parte de la práctica en el contexto de la problemática de un investigación determinada.


*Figura 5.1. Investigación artística en tecnología musical en el caso del proyecto de live coding SonoTexto.*

Con lo anterior no quiero proponer que la investigación artística en tecnología musical, en el ámbito del live coding, sea un campo de desarrollo tecnológico *per se* sino uno en el que la

creación artística y el desarrollo tecnológico se vinculan en contextos artísticos específicos. Así, la práctica artística con lenguajes de programación forma su vocabulario y sus formas de expresión con base en objetos computacionales vinculados a, y desarrollados para, dicha práctica artística y viceversa. Una forma imbricada que ofrece posturas, decisiones, intenciones y perspectivas para producir conocimiento.

## 5.4 El objeto computacional abierto

Para estudiar la relación entre práctica artística y desarrollo tecnológico en el live coding caractericé estas partes como los modos de programación exploratoria y prescriptiva. A partir de esa caracterización accedí a desarrollar la serie de objetos computacionales SonoTexto y a realizar live coding con ellos. La llegada a este vínculo fue a través de las ideas de exploración y prescripción presentes en los trabajos de Carolina Di Próspero y Ursula Franklin. A partir de esto pude crear una unidad de análisis expresada en una gráfica que vincula la abstracción con las formas de programación propuestas (véase § 3.3). El trabajo se centró en construir el andamiaje teórico y conceptual para analizar la relación entre la práctica artística y el desarrollo tecnológico en el live coding que me permitió llegar a diseñar objetos computacionales abiertos para hacer live coding desde una idea musical.

La problemática de la que partí fue la barrera entre conocimientos y modos de hacer de la programación exploratoria y de la programación prescriptiva. El discurso del live coding suele colocar ambas aproximaciones en la misma arena con el deseo de borrar la barrera que separa a quien crea música de quien desarrolla el software. La apertura del código fuente, el acceso al conocimiento colectivo y la abstracción alta de los lenguajes de programación son un primer paso para desdibujar tal barrera. Sin embargo, la diferencia de conocimientos en los modos de programación no se resuelve por el hecho de que una tecnología se declare abierta. Para franquear tal barrera se necesita reconocer las partes y, posteriormente, trasponer ideas, conceptos, prácticas y modos de hacer entre ellas aprovechando la apertura y la abstracción que permiten desglosar los mecanismos y los saberes del, y sobre el, software.

Esta separación no surgió en la escena del live coding, se observa ya desde la división de especialidades en los inicios del arte computacional. Por ejemplo, Raisa Reichardt (1969) comenta que la aparición de la computadora en el arte en los años 1960 se vio como una posibilidad de colaboración entre artistas y científicos desde sus campos de especialización. Aquí Reichardt expresa una división por especialidad de la época entre quienes saben usar las computadoras porque tienen acceso a ellas y quienes ven la posibilidad de expresarse de manera creativa pero dependen de quienes las programan.

La división del trabajo expresada por Reichardt que se extiende a la colaboración entre artista e ingeniero de software o científico de la computación, es una forma de trabajo recurrente en el arte computacional en la que el artista propone y el ingeniero resuelve. Sin embargo, el live coding muestra que esa división de especialidades se unen en la misma persona quien diseña el software y lo usa para crear música o imagen (véase § 2.4). Esto no sugiere un trabajo solitario pues hay una colaboración que se distribuye a partir del acceso al conocimiento colectivo, ya sea por la investigación académica o por la ayuda en foros.

Pero, ¿qué sucede con la relación entre música, sonido y código fuente? ¿es posible lograr que la idea musical y sonora se expresen con el diseño y desarrollo de una pieza de software? La persona que emprende un proyecto artístico parte de ideas, necesidades de expresión y de la imaginación. Una de las posibilidades para desarrollar esa idea es bajo los lineamientos de un lenguaje de programación. Al programar un objeto computacional, de manera prescriptiva, la idea artística se formaliza a través de un conocimiento distinto al que se requiere para hacer música. El sonido imaginado o pensado, como es descrito por Beatriz Ferreyra (2019) en el ámbito de la música electroacústica, en el live coding se enfrenta a una traducción a código fuente para su control. La grabación de sonido nos enfrenta a escuchar, pensar y, en el caso del live coding, a codificar el sonido en una situación de improvisación. En esta situación, el sonido imaginado se convierte en datos, la idea artística se adapta al lenguaje de programación y emerge una vez terminada la escritura de código fuente. La idea artística se traslada a la implementación tecnológica del software.

El live coding está colocado en la frontera entre formas de aproximación al sonido, en la que domina la tendencia del sonido imaginado a través del lenguaje de programación y no como

elemento separado de su fuente sonora. En el live coding, el momento de escucha de la creación musical ocurre después de que el código entrega su resultado. Sin embargo, hay un momento previo en el que pensamos cómo controlar el sonido e imaginamos cómo puede sonar el resultado. Ese momento comienza desde el diseño del objeto computacional al que le añadimos las funciones de un sonido previamente imaginado. Esto también ocurre en el momento de hacer live coding en el que tal reflexión sucede de manera inmediata bajo las estructuras y funciones de un lenguaje de programación que hemos memorizado. No sabemos cómo va a sonar pero tenemos una idea de cómo se va a comportar. El resultado lo escucharemos hasta el momento de la presentación en vivo.

El desarrollo tecnológico de SonoTexto ofrece una pieza de software u objeto computacional que parte de una idea artística cargada del bagaje personal y se enmarca en los límites del conocimiento para desarrollarla y no en los límites del sonido imaginado. El resultado es un pequeño mecanismo de grabación que me permite abrir el micrófono en una presentación de live coding para capturar sonido y después, con las herramientas convencionales de SuperCollider, hacer live coding. Además, este mecanismo de grabación se extiende mediante otras clases que permiten la organización de sonido (datos) y otro para colocarlo en el tiempo (patrón). En conjunto es un objeto computacional abierto para hacer live coding compuesto por tres clases que graban, organizan y secuencian sonido.

Aquí el dualismo de los conceptos que Schaeffer (1966/2008) detectó en la definición de la música concreta logra la transversalidad entre las ideas de la música y las ciencias de la computación en la in-corporación de código fuente que plantea Baalman (2015). Intentamos escribir código fuente a prueba y error para llegar al resultado musical y artístico en una actividad en la que hemos arraigado nuestras habilidades. Es ahí donde detecto el pensamiento computacional expresado en esta práctica artística. Una forma de pensar referida en la pregunta de cómo pensamos con, y a través de, la tecnología que plantea Magnusson (2019), el argumento de que pensamos con, a través de, y junto a, los medios propuesto por Hayles (2012), y la proposición de Hui (2010) acerca de la forma de ver el mundo a través de lo computacional en el giro que ello conlleva. El objeto computacional abierto que deriva de esta investigación transita

por este pensamiento, el sonido se traslada al código fuente para imaginar una práctica musical a través de lo computacional.

# Consideraciones finales

La programación prescriptiva coloca idiosincrasia, preferencias musicales, necesidades, deseos, intenciones y afiliaciones dentro del software que desarrollamos y la música que creamos con él. Ello apunta a construir mecanismos computacionales que nos permiten expresarnos mediante la programación exploratoria de la creación artística y musical del live coding. Esta práctica no aparece y desaparece de la hoja en blanco, está construida sobre mecanismos computacionales que resuelven la expresión artística con lenguajes de programación en los que persiste el live coding. Desde esta perspectiva, la práctica artística comienza desde la conceptualización y construcción del objeto computacional y se materializa en lo que podemos expresar con él durante la práctica artística. Tal relación es un desplazamiento necesario entre el arte y las ciencias de la computación que hace posible un diálogo más amplio sobre las implicaciones creativas del software en el arte computacional. Esto nos permite cuestionar la dependencia tecnológica al software al que estamos afiliados y, en colaboración con sus desarrolladores y desarrolladoras, usuarios y usuarias, pensar formas de diseñar modos de expresión con objetos computacionales abiertos.

Para concluir este trabajo me remito a las preguntas, la propuesta y la hipótesis inicial de la investigación enunciadas en la introducción. Antes recuerdo los ejes del trabajo: la práctica artística caracterizada como la programación exploratoria en el live coding, el desarrollo tecnológico como la programación prescriptiva de objetos computacionales y la investigación artística en tecnología musical. Las preguntas iniciales fueron: ¿cómo se transforma la práctica artística del live coding en el desplazamiento de la programación exploratoria a la programación prescriptiva? ¿es suficiente señalar estas formas de programación para desdibujar la supuesta barrera entre práctica artística y desarrollo tecnológico en el ámbito del live coding? ¿cómo se relacionan ambas partes dentro de una investigación artística? Mismas que abordé desde la siguiente propuesta: “el objeto computacional que resulta del desarrollo tecnológico forma parte de la práctica artística y dicha relación puede informar y ser informada en el contexto de la práctica investigativa. En este sentido, la finalidad del desarrollo tecnológico no es llegar a un

objeto terminado autónomo sino a un objeto que actúa y se transforma junto a la práctica artística en el marco de la investigación". Con ello, me propuse mostrar que "la práctica artística del live coding sufre una transformación en la medida que accede a diversas capas del desarrollo tecnológico que están vinculadas a los niveles de abstracción de los lenguajes de programación". Donde, "la apertura del código fuente es la vía que permite un desplazamiento de ida y vuelta entre la práctica artística y el desarrollo tecnológico, lo que se observa de manera concreta entre las formas de programar exploratoria y prescriptiva del live coding".

De esta manera, el proceso de creación musical ocurre bajo una dinámica de prueba y error en el que "descubrimos" funciones de un lenguaje de programación que vamos incorporando a nuestra práctica artística, comúnmente a través de la memorización de líneas de código. De esta manera, la programación exploratoria opera bajo las posibilidades del lenguaje de programación guiadas por el bagaje personal de cada live coder, tanto musical como informático. Esta dinámica nos confronta a nuestros modos de hacer artísticos que, a su vez, plantean problemas computacionales. En este punto podemos decidir si damos paso, o no, a la implementación de las formas de creación que hemos descubierto. En tal caso, nuestros hallazgos creativos encontrados en la programación exploratoria reflejan la transformación que sufre la práctica artística cuando esta se involucra con el desarrollo tecnológico de sus objetos computacionales. Esto ocurre en el proceso de sistematización y estructuración de la práctica artística cuando desarrollamos tales objetos computacionales. En ese proceso, el desarrollo tecnológico también sufre una transformación en la forma de plantear los problemas que vienen de la práctica artística.

Además, en el caso de esta investigación, la práctica artística y el desarrollo tecnológico no solo se transformaron a través de su relación intrínseca, también lo hicieron debido a la circunstancias derivadas de la pandemia, lo que implicó la implementación de políticas institucionales y formas de organización social que han buscado dar continuidad a la vida académica y cultural durante este periodo. Con base en lo anterior es posible afirmar que las transformaciones que sufre la práctica artística y el desarrollo tecnológico también son de tipo social.

La respuesta a la segunda pregunta es simple si solo señalamos las partes, pues en tal caso la barrera sigue ahí. En cambio, esta barrera se puede comenzar a desdibujar mediante un movimiento de transposición entre conocimientos de la práctica artística y el desarrollo tecnológico que nos permite operar en ambos lados. Así que, señalar las partes no basta para desdibujarla, hay que empujar el movimiento de ida y vuelta. Aquí cabe señalar que, al dividir la programación por sus metodologías se crea una separación entre el objeto computacional como resultado y la práctica artística como proceso que, ciertamente permite visibilizar la barrera. Sin embargo, existe un riesgo si tratamos cada parte de forma independiente pues esto acaba reforzando la dicotomía e impide desdibujar la frontera. Es aquí donde la relación entre práctica artística y desarrollo tecnológico encuentra una alternativa en la transposición entre conocimientos de la música, las artes y las ciencias de la computación. Esto en la medida que la investigación y el desarrollo tecnológico del live coding sucedan en contextos de tecnología y conocimiento abiertos.

Teniendo en cuenta lo anterior, SonoTexto pudo indagar en la programación exploratoria y en la prescriptiva para llegar a una práctica artística vinculada con su desarrollo tecnológico. Para ello, fue necesario cambiar la perspectiva de programación al implementar las ideas artísticas en un lenguaje de programación bajo la dinámica de resolución de problemas computacionales, orientados a la música, propia del desarrollo tecnológico. Lo anterior sucedió bajo una lógica de *resolver para expresar*. Con ello, la idea artística se incorpora a los recursos del lenguaje de programación y, posteriormente, emerge con el uso de este. Lo anterior posibilita dejar de ver dos partes separadas que, por un lado, crean la sensación de una práctica artística dependiente del desarrollo tecnológico y, por otro, la de un desarrollo tecnológico que resuelve problemas técnicos desvinculado de su contexto artístico.

La aproximación a diluir la barrea entre el artista que programa y el programador que desarrolla objetos computacionales no deja de ser problemática. Si bien se puede “trascender” para incluir formas de pensamiento, de creación y de expresión en un lenguaje de programación determinado, lo que se inscribe en él también es una forma de pensamiento musical con reglas y protocolos propios. En otras palabras, lo anterior posibilita construir formas de expresión musical dentro de los lenguajes de programación pero, a través de esa construcción, se introducen formas

de pensamiento musical. Esto forma un doble juego de lo prescriptivo en el desarrollo de tecnología musical informática que guía cómo hacer software pero también cómo hacer música. Lo anterior ocurre desde las reglas del lenguaje de programación, así como desde las reglas de la creación musical implementadas en él. En este sentido, la programación exploratoria también es prescriptiva.

Respecto a la tercera pregunta, la forma en que la práctica artística y el desarrollo tecnológico se vinculan en la investigación sucede desde el momento en que cuestionamos su relación. En la práctica del live coding ambas partes están entrelazadas y utilizan una tecnología informática donde no está clara la división entre ellas. La investigación ayuda a reconocer las partes y las separa para analizarlas. Esta separación es intencional y dirigida por el planteamiento de cada investigación. En este caso, el hecho de reconocer dos formas de programar posibilitó enunciar tal relación y activarla bajo la idea de transponer los conocimientos artísticos y tecnológicos involucrados en ella.

En el momento de desarrollar un objeto computacional lo anterior se hace evidente, puesto que al seguir una serie de pasos estructurados en la programación afloran no solo las diferencias de conocimiento tecnológico sino las restricciones de acceso que seguimos enfrentando a pesar de la apertura del código fuente y del conocimiento. Esto en parte es consecuencia de una tradición del pensamiento computacional que se refleja en la historia del arte computacional y la música por computadora. Lo anterior está imbricado en los estratos sobre los que están construidas las tecnologías con las que realizamos nuestra práctica artística computacional. En consecuencia los objetos computacionales con los que realizamos live coding tienen inscrita esa línea histórica en su desarrollo tecnológico.

La invitación abierta para apropiarnos de los lenguajes de programación libres sin duda nos permite crear música, pero como se ha planteado en este trabajo, esta invitación debería involucrar tanto el *saber cómo* programar con ellos así como el conocimiento para diseñar y construir las funciones y estructuras que permiten la creación artística. Esto implica participar en las discusiones de una comunidad de live coding que desea construir su práctica bajo principios distintos a los establecidos por las jerarquías de la historia del arte y la computación. Es aquí donde radica el reto de dialogar desde nuestras experiencias, ideas, modos de hacer y posturas.

Hasta aquí llega mi propuesta para ingresar y colaborar a desdibujar una barrera que separa una práctica de manera más compleja que la simple división entre conocimientos tecnológicos y artísticos. Con este trabajo, espero contribuir al arte computacional a través de la práctica del live coding en México y generar un incentivo para quienes quieren crear, desarrollar e investigar en el ámbito de la tecnología musical. De la misma manera, deseo alentar el desarrollo de software desde la práctica artística y la propuesta de estrategias metodológicas de investigación en el campo de la música y el arte computacionales. A partir de lo anterior, espero lograr planteamientos educativos y estéticos con perspectivas tecnológicas, sociales y culturales que apunten a la ruptura de la dependencia tecnológica que prescribe la práctica artística y sus narrativas. En este sentido, planteo abrir cajas de abstracción para entender mejor la tecnología en relación a la cultura, pero sobre todo para entendernos en un acto de comunicación humana.

# Referencias


Abelson, H., Sussman, G. J., y Sussman, J. (1996). *Structure and interpretation of computer programs* (2a de.). MIT Press.

Arkbauer. (n.d). Software development life-cycle (SDLC). *Arkbauer*. https://arkbauer.com/blog/software-development-life-cycle-sdlc/

Azor, I., Grijalva Maza, L. F., y Gómez Rossi, A. A. R. (Eds.). (2016). *Más allá del texto: Cultura digital y nuevas epistemologías* (Primera edición). Universidad de las Américas, Puebla ; Editorial Itaca.

Baalman, M. (2015). Ebodiment of code. En A. McLean, T. Magnusson, N. Kia, S. Knotts, y J. Armitage (Eds.), *Proceedings of the First International Conference on Live Coding* (pp. 35–40). University of Leeds.

Babini, D., y Rovelli, L. (2020). *Tendencias recientes en las políticas científicas de ciencia abierta y acceso abierto en Iberoamérica*. Buenos Aires: CLACSO y Fundación Carolina.

Barragán, H. (2007). Software: ¿arte? En La Ferla, J. (Ed.). *El medio es el diseño audiovisual* (pp. 583–587). Manizales: Universidad de Caldas.

Bernardo, F., Kiefer, Ch. y Magnusson, T. (2020). Designing for a Pluralist and User-Friendly Live Code Language Ecosystem with Sema. En *Memorias del ICLC 2020* (pp. 41–57). Limerick, Ireland: University of Limerick. http://doi.org/10.5281/zenodo.3939228

Berry, D. M., y Dieter, M. (2015). Thinking Postdigital Aesthetics: Art, Computation and Design. En Berry, D. M., y Dieter, M. (Eds.), *Postdigital Aesthetics: Art, computation and design* (pp. 1–11). Palgrave Macmillan UK. https://doi.org/10.1057/9781137437204_1

Berry, D. M. (2011). *The philosophy of software: Code and mediation in the digital age*. Basingstoke: Palgrave Macmillan.

Berry, D. M. (2004). The contestation of code: A preliminary investigation into the discourse of the free/libre and open source movements. *Critical Discourse Studies*, *1*(1), 65–89. https://doi.org/10.1080/17405900410001674524

Borgdorff, H. (2012). *The conflict of the faculties: Perspectives on artistic research and academia*. Leiden University Press.

Bolanakis, D. E., Evangelakis, G. A., Glavas, E., y Kotsis, K. T. (2011). A teaching approach for bridging the gap between low-level and high-level programming using assembly language

learning for small microcontrollers. *Computer Applications in Engineering Education*, *19*(3), 525-537. https://doi.org/10.1002/cae.20333

Brookshear, J. G. (2012). *Introducción a la computación (11a. Ed.).* Pearson Educación.

Brown, A. R. (2007). *Computers in music education: Amplifying musicality*. Nueva York: Routledge.

Bovermann, T., & Griffiths, D. (2014). Computation as Material in Live Coding. *Computer Music Journal*, *38*(1), 40-53. https://doi.org/10.1162/COMJ_a_00228

Candy, L., Edmonds, E. A., y Poltronieri, F. (2018). *Explorations in art and technology* (Second edition). Londres: Springer.

CaosBox [Software de computadora]. (2014). Recuperado de https://github.com/josecaos/caosbox

Carvalhais, M., y Cardoso, P. (2018). Empathy In The Ergodic Experience Of Computational Aesthetics. En R. Adebayo, I. Farouk, S. Jones, y M. Rapeane-Mathonsi (Eds.), *ISEA 2018 Intersections* (pp. 220-226). https://doi.org/10.5281/ZENODO.1303348

Cassany, D. (2005). Investigaciones y propuestas sobre literacidad actual: multiliteracidad, Internet y criticidad. En Véliz de Vos, M. (Ed.), *Congreso Nacional de Cátedra UNESCO para la Lectura y Escritura* (pp. 1-10). Concepción.

CMM. (s.f.). *Centro Multimedia*. http://cmm.cenart.gob.mx/acercaDe

Chapman, O., y Sawchuck, K. (2012). Research-Creation: Intervention, Analysis and “Family Resemblances”. *Canadian Journal of Communication*, *37*(2012), 5-26.

Chiantore, L., Domínguez, A. y Martínez, S. (2016). *Escribir sobre música*. Valencia: Musikeon.

Chisnall, D. (2018). C Is Not a Low-level Language. Your computer is not a fast PDP-11. *ACM Queue*, *16*(2). https://doi.org/10.1145/3212477.3212479

Chun, W. H. K. (2008). On “Sourcery,” or Code as Fetish. *Configurations*, *16*(3), 299–324. https://doi.org/10.1353/con.0.0064

Collins, N., McLean, A., Rohrhuber, J., y Ward, A. (2003). Live coding in laptop performance. *Organised Sound*, *8*(3), 321–330. https://doi.org/10.1017/S135577180300030X

Cox, G. (2015). What does live coding know? En McLean, A., Magnusson, T., Ng K., Knotts, S. y Armitage, J. (Eds.), *International Conference on Live Coding* (pp. 93-97). Leeds.

Cox, G., y McLean, A. (2013). *Speaking code: Coding as aesthetic and political expression*. Cambridge, Mass: The MIT Press.

Cramer, F. (2005). Words made flesh: Code, Culture, Imagination. Rotterdam: Piet Zwart Institute.

Crowther, P. (2019). *Digital art, aesthetic creation: The birth of a medium*. Nueva York: Routledge.

Dean, R. T. y McLean, A. (Eds.) (2018). *The Oxford Handbook of Algorithmic Music*. Oxford: Oxford University Press.

Di Próspero, C. (2015). Live coding. Arte computacional en proceso. *Contenido. Arte, Cultura y Ciencias Sociales*, *5*(2015), 44–62.

Di Próspero, C. (2017). Escrito en el cuerpo. Nuevas performances tecnoartísticas. *Artelogie*, *11*. https://doi.org/10.4000/artelogie.1672

Di Próspero, C. (2019). Subjetividades tecnoartísticas: intervención imaginativa. *De signos y sentidos*, (20), 6-29.

Doornbusch, P. [pauldoornbusch]. (2009, 28 de septiembre). *Pearcey Interview about CSIRAC Music* [Video]. YouTube. https://youtu.be/slr75sLhOCs

Franklin, U. (1990). *The real world of technology*. Toronto: CBC Enterprises.

Free Software Foundation (2022). *About*. Recuperado de https://www.fsf.org/about/

Edmond, J. (Ed.) 2020. *Digital Technology and the Practices of Humanities Research.* Open Book Publishers. Retrieved from http://books.openedition.org/obp/11899

Ferreyra B. [Q-O2 workspace]. (14 de octubre de 2019). *Beatriz Ferreyra "Identificaction of sound characteristics"* [Video]. YouTube. https://youtu.be/dWEDcq56D_Q

Fuller, M. (2008). *Software studies: A lexicon*. Cambridge: MIT Press.

García, J. D. y Silva Treviño, P. (Eds.) (2022). *Algoritmos arruinados: Perspectivas situadas sobre tecnología musical*. Ciudad de México: UNAM.

García, J. D. (2021). Tecnología libre para una música libre: ¿es posible construir un sistema de producción musical alternativo? *Ideas Sónicas*, *23*, 29-45.

GNU. (2021). *What is free software?* https://www.gnu.org/philosophy/free-sw.en.html#four-freedoms

Güerca Torres, R., Blásquez Martínez, L. I. y López Moreno, I. (2016). *Guía para la investigación cualitativa: etnografía, estudio de caso e historia de vida*. Ciudad de México: UAM.

Hayles, K. (2012). *How we think: Digital media and contemporary technogenesis*. Chicago; London: The University of Chicago Press.

Harkins, J. [jamshark70](2020, 17 de febrero). *Computer ensemble remote-leraning methodologies* [Post en foro en línea]. Toplap forum. https://forum.toplap.org/t/computer-ensemble-remote-learning-methodologies/1106

Herrera Lima, M. (2019). Historia del arte y estética: encuentros y desencuentros. En Richterich, K. (Ed.), *Historia del arte y estética, nudos y tramas: XXXIX Coloquio Internacional de Historia del Arte*, (pp. 285-300). UNAM.

Herrera Machuca, M., Lobato Cardoso, J. A., Torres Cerro, J. A., y Lomelí Bravo, F. J. (2016). Live coding for all: Three creative approaches to live coding for non-programmers. *International Journal of Performance Arts and Digital Media*, *12*(2), 187–194. https://doi.org/10.1080/14794713.2016.1227598

HFBK. (2004). *Changing Grammars*. https://swiki.hfbk-hamburg.de/MusicTechnology/609

Himanen, P. (2001). *The hacker ethic and the spirit of the information age*. Nueva York: Random House Trade Paperbacks.

Higgins, B H. y Kahn, D. (Eds.) (2012). *Mainframe experimentalism: Early Computing and the Foundations of the Digital Arts*. Berkeley: University of California Press.

Holmes, S. (2016). "Can We Name the Tools?" Ontologies of Code, Speculative Techné and Rethorical Cocealment. *Computational Culture, 5*. Recuperado de: http://computationalculture.net/can-we-name-the-tools-ontologies-of-code-speculative-techne-and-rhetorical-concealment/

Hui, Y. (2010). The Computational Turn, or, a New Weltbild. *Junctures*, (13), 41-51.

Hydra [Software de computadora]. (2018). Recuperado de https://github.com/ojack/hydra

INSTRUMENT [Software de computadora]. (2017). Recuperado de https://github.com/punksnotdev/INSTRUMENT

Impett, J. (2016). *Music, Thought and Technology*. https://orpheusinstituut.be/en/projects/music-thought-and-technology

Imppet, J., De Assis, P., Beghin, T., Laws, C. y Vaes, L. (s.f.). *Artistic Research in Music – an introduction*. [MOOC]. edX. https://www.edx.org/es/course/artistic-research-in-music-an-introduction

International Conference on Live Coding. (2021, 22 noviembre). *Hybrid Live Coding Interfaces 2021 [2] Journaling* [Video]. YouTube. https://youtu.be/gixyLo8JVU0

Jordá, S. (2005). *Digital Lutherie: Crafting musical computers for new musics' performance and improvosation* (PhD Thesis). Departament de Tecnologia, Universitat Pompeu Fabra.

Joyanes Aguilar, L. (2020). *Fundamentos de programación: Algoritmos, estructuras de datos y objetos*. McGraw-Hill Interamericana.

Keislar, D. (2012). A Historical View of Computer Music Technology. En Dean, R. T. (Ed.). *The Oxford Hanbook of Computer Music*. Oxford: Oxford University Press. https://doi.org/10.1093/oxfordhb/9780199792030.013.0002

Knotts, S. y Paz, I. (2021). Live Coding and Machine Learning is Dangerous: Show us your Algorithms. *Proceedings of the International Conference on Live Coding*. Valdivia, Chile. https://doi.org/10.5281/zenodo.5888356

Knotts, S. y Armitage, J. [Yorkshire Sound Women Network]. (2015, 17 de enero). *Live coding – Yorkshire Sound Women Network & AHRC Live Coding Research Network* [Video]. YouTube. https://youtu.be/RpzEzUCgVoQ

LaBelle, B. (2008). *Background Noise: Prespectives on Sound Art*. Nueva York: Continuum.

Larrieu, M. (2019). A Consideration of the Code of Computer Music as Writing, and Some Thinking on Analytical Theories. *Organised Sound*, *24*(3), 319–328. https://doi.org/10.1017/S1355771819000384

Lawson, S. y Smith, R. R. (2019). What Am I Looking At?: An Approach to Describing the Projected Image in Live-Coding Performance. *Proceedings of the International Conference on Live Coding*. Madrid, España: MediaLab Prado.

Lee, S. W. (2019). Show Them my Screen: Mirroring a Laptop Screen as an Expressive and Communicative Means in Computer Music.

Levy, S. (2010). *Hackers: Heroes of the computer revolution*. Sebastapol: O'Reilly Media.

López-Cano, R. y San Cristóbal Opazo, U. (2014). *Investigación artística en música: Problemas, métodos, experiencias y modelos*. Barcelona: FONCA/ESMUC.

Magnusson, T. (2021). *Scoring Sound: Creative Music Coding With SuperCollider*. Victoria: Leanpub.

Magnusson, T. (2019). *Sonic writing: Technologies of material, symbolic and signal inscriptions*. Nueva York: Bloomsbury Academic.

ManifestoDraft (s.f). En TOPLAP. https://toplap.org/wiki/ManifestoDraft

Marino, M. C. (2020). *Critical code studies: Initial methods*. Cambridge: The MIT Press.

Matthews, W. (2012). *Improvisando. La libre creación musical*. Madrid: Turner Música.

McLean, A. y Wiggins, G. (2012). Computer Programming in the Creative Arts. En J. McCormack y M. d'Inverno (Eds.), *Computers and Creativity* (pp. 235-252). https://doi.org/10.1007/978-3-642-31727-9

McLean, A. (2011). *Artist-Programmers and Programming Languages for the Arts* (PhD Thesis). Department of Computing, Goldsmiths, University of London.

MDN web docs. (2005). Classes. *MDN web docs*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

Mineault P. J. (2021). *The Good Research Code Handbook*. Zenodo. doi:10.5281/zenodo.5796873

MIRLCa [Software de computadora]. (2020). Recuperado de https://github.com/mirlca

Monreal Ramírez, J. F. (2020). *Máquinas para descomponer la mirada: Estudios sobre la historia de las artes electrónicas y digitales en México* (Primera edición). Ciudad de México: UAM y Juan Pablos Editor.

Mori, G. (2015). Analysing Live Coding with Ethnographic Approach—A New Perspective. En McLean, A., Magnusson, T., Ng K., Knotts, S. y Armitage, J. (Eds.), *International Conference on Live Coding*. Recuperado de https://doi.org/10.5281/ZENODO.19343

Nierhaus, G. (2009). *Algorithmic Composition: Paradigms of Automated Music Generation*. Viena: Springer-Verlag.

Ocelotl, E., Del Angel, L. N., y Teixido, M. (2018). Saboritmico: A Report from the Dance Floor in Mexico. *Dancecult*, *10*(1). https://doi.org/10.12801/1947-5403.2018.10.01.11

Open Source Initiative. (2018). *History of the OSI*. Recuperado de https://opensource.org/history

Parra Cancino, J., y Impett, J. (2020). Thought, Technology and Performance: Lessons from the Future. *Music & Practice*, *6*, 1 – 9. https://doi.org/10.32063/0604

Peña Pimentel, M. y Barrón, Tovar, J. F. [Instituto de Investigaciones Antropológicas, UNAM]. (2020, 22 de enero). *Sesión 8 Humanidades Digitales* [Video]. YouTube. https://youtu.be/qnvNPUt0GcE

Piranha Lab (2019, julio 23). *Conversatorio PiranhaLab 23 jul 2019* [Video]. YouTube. https://youtu.be/6ajFAuQQZpU

Pitchard, H., Snodgrass, E. y Tizlik-Carver, M. (2018). *Executing Practices*. Aarhus: Open Humanities Press.

Quintero Álvarez Icaza, L. (2013). Producción estética con tecnología: un vestigio de resistencia. En Saldaña Hernández, J. C. y Villagómez Oviedo, C. (Eds.). *Arte en la era digital: estudios críticos y nuevos lindes*. Guanajuato: Universidad de Guanajuato.

Raymond, E. S. (2004). *The Jargon File, versión 4.4.8*. http://www.catb.org/~esr/jargon/index.html

Reichardt, J. (2005, septiembre 28 – octubre 1). *The computer in art*. [Presentación en congreso]. Refresh! 1st International Conference on the Histories of Art, Science and Technology, Banff, Alberta, Canadá.

Reichardt, J. (1969). *Cybernetic Serendipity: The Computer and the Arts*. New York; Washington: Frederik A. Praeger, Publishers.

Reppel, N. (2020). The Megra System - Small Data Music Composition and Live Coding Performance. En *Memorias del ICLC 2020* (pp. 95–104). Limerick, Ireland: University of Limerick. http://doi.org/10.5281/zenodo.3939154

Rodríguez Peña, D. A. [Cultura UMSNH]. (9 de febrero, 2022). *Hablemos de mundos | Transformación digital en la enseñanza de las artes* [Video]. YouTube. https://youtu.be/cnMCXaYYvCA

Rohrhuber, J. y De Campo, A. (2009). Improvising Formalisation – Conversational Programming and Live Coding. En Assayag, G., y Gerzso, A. (Eds.). *New Computational Paradigms for Computer Music*. Paris: Delatour France.

Sáizar, C. (2009). Presentación. En Lemus, A. (ed.). *Centro Multimedia Centro Nacional de las Artes: Quince años*. Ciudad de México: Centro Nacional de las Artes.

Sastre, P. (2021). *Manifiestos sobre el arte y la red. 1990-1999*. Madrid: Exit.

Schaeffer, P., y Cabezón de Diego, A. (2008). *Tratado de los objetos musicales*. Madrid: Alianza. (Publicado originalmente en 1966)

Schwab, M. (Ed.). (2018). *Transpositions: Aesthetico-epistemic operators in artistic research*. Leuven: Leuven University Press.

Seis8s [Software de computadora]. (2020). Recuperado de https://seis8s.org/

Sema [Software de computadora]. (2019). Recuperado de https://github.com/mimic-sussex/sema

Shanken, E. (2004). *Historizing Art and Technology: Forging a Method and Firing a Canon*. Media Art Histories Archive. http://95.216.75.113:8080/xmlui/handle/123456789/297

Shulman, A. (1999). The Style Bible: an A-Z of global youth culture. Londres: Methuen.

Sicchio, K. (2019). *Programming paradigms for the human interpreter*. International Conference on Live Coding 2019, Madrid, España.

Small, C. (1998). *Musicking: The meanings of performing and listening*. University Press of New England.

Soon, W. y Cox, G. (2020). *Aesthetic Programming: A Handbook of Software Studies*. Aarhus: Open Humanities Press.

Sorensen, A., Swift, B., y Riddell, A. (2014). The Many Meanings of Live Coding. *Computer Music Journal*, *38*(1), 65–76.

Soria Guzmán, M. I. (2020). Mujeres hacker, saber-hacer y código abierto: Tejiendo el sueño hackfeminista. *LiminaR Estudios Sociales y Humanísticos*, *19*(1), 57–74. https://doi.org/10.29043/liminar.v19i1.806

Sormani, P., Carbone, G., y Gisler, P. (Eds.). (2019). *Practicing art/science: Experiments in an emerging field*. Londres y Nueva York: Routledge.

Stewart, J., Lawson, S., Hodnick, M., y Gold, B. (2020). Cibo v2: Realtime Livecoding A.I. Agent. En *Memorias del ICLC 2020* (pp. 20–31). Limerick, Ireland: University of Limerick. http://doi.org/10.5281/zenodo.3939174

SuperCollider 3.12.0 [Software de computdora]. (2021). Recuperado de https://github.com/supercollider/supercollider/releases/tag/Version-3.12.0

Tello, A. M. (Ed.). (2020). *Tecnología, política y algoritmos en América Latina*. Viña del Mar: CENTALES Ediciones LTDA.

Temkin, D. (2017). Language Without Code: Intentionally Unusable, Uncomputable, or Conceptual Programming Languages. *Journal of Science and Technology of the Arts*, *9*(3), 83. https://doi.org/10.7559/citarj.v9i3.432

Thompson, R. y Mukhopadhyay, T. P. (2021). Digital arts in Latin America: A report on the archival history of intersections in art and technology in Latin America. *Digital Scholarship in the Humanities*, *36*(1), i113–i123. https://doi.org/10.1093/llc/fqaa046

UNAM Posgrado Música (s.f.). *Campos de conocimiento*. https://www.posgrado.unam.mx/musica/div/campos/areas.html

Vallverdú, J. (s.f.). *Humanidades digitales* [MOOC]. Coursera. https://www.coursera.org/learn/humanidades-digitales

Verostko, R. (2003). Epigenetic Art Revisited: software as geonotype. [Artículo en línea] Recuperado de http://www.verostko.com/archive/writings/epigen-art-revisited.html

Villaseñor Ramírez, H. y Paz, I. (2020). Live coding from scratch: The cases of practice in Mexico City and Barcelona. *Proceedings of the International Conference on Live Coding*, 59-68. https://doi.org/10.5281/zenodo.3939206

Wakefield, G., y Roberts, C. (2017). A Virtual Machine for Live Coding Language Design. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 275–278. Recuperado de http://www.nime.org/proceedings/2017/nime2017_paper0052.pdf

Ward, A., Rohrhuber, J., Olofsson, F., McLean, A., Griffiths, D., Collins, N., y Alexander, A. (2004). Live Algorithm Programming and a Temporary Organisation for its Promotion. En O. Goriunova y A. Shulgin (Eds.), *Read_me: Software Art & Cultures* (pp. 243-261). Aarhus: Digital Aesthetics Research Centre, University of Aarhus.

Wang, G. y Cook, P. (2004). On-the-fly Prgramming: Using Code as an Expressive Musical Instrument. En *Proceedings of the International Conference on New Interfaces for Musical Expression*, 138-143. https://doi.org/10.5281/zenodo.1176682

Xambó, A., Lerch, A. y Freeman, J. (2019). Music Information Retrieval in Live Coding: A Theoretical Framework. *Computer Music Journal*, 42(4), 9-25.

Xambó, A. (2016, Noviembre 12). The top 10 Most-cited Papers in Music Technology. *Anna Xambó PhD*. http://annaxambo.me/blog/research/2016/11/12/top-10-most-cited-papers-in-music-technology.html

Young, J. (2015). Imaginary Workscapes: Creative Practice and Research through Electroacoustic Composition. En Doğantan, M. (Ed.). *Artistic practice as research in music: Theory, criticism, practice*. Farnham: Ashgate.

Zećo, M. (2021). Listening and Recording In Situ: Entanglement in the sociopolitical context of place. *Organised Sound*, *26*(2), 284–290. https://doi.org/10.1017/S1355771821000327

# Anexos

## A. Código Fuente

El código fuente de las clases está colocado bajo la propuesta que hace Mark Marino en su libro *Critical Code Studies* (2020) para leerlo. Esta propuesta consiste en escribir la información de los archivos al inicio de su exposición y después desplegar el código fuente, incluyendo el número de línea y los comentarios, como están escritos en su fuente original. En el caso de este trabajo el código fue recuperado de mi perfil de GitHub (https://github.com/hvillase), el cual está sincronizado con el código fuente que permanece en la computadora donde lo escribo. La numeración de las líneas la agregué con el procesador de texto. Hice una excepción con la tipografía utilizada para el código fuente por lo que en este apartado utilizo la misma fuente que el cuerpo del texto.

### A.1 SonoTexto

**Archivos:** sonotexto.sc, sonotexto-synths.scd

**Lenguaje de programación:** sclang SuperCollider

**Desarrollado:** 2019 – 2021

**Autor:** Hernani Villaseñor

**Plataforma:** GNU/Linux, Windows, Mac OS, Raspberry Pi OS

**Repositorio:** https://github.com/hvillase/sonotexto

**Código fuente del archivo sonotexto.sc**

```
1. // SonoTexto class
2. // This is a class to record, play and save sounds of 5 and 10 seconds duration
3. // It was developed for a live coding performance but can be used to compose music
4.
5. SonoTexto {
6.
7.      // the variable st means sonotexto, it is intended to make a Dictionary
8.      classvar <st;
9.
10.     // the class method *boot check if there is a folder to write sounds and executes the
        document called sonotexto-synths.scd
11.     *boot {
12.
13.      // this alerts when the sound folder sonotexto is not in the Recordings path.
14.      if (File.existsCaseSensitive(Platform.recordingsDir ++ "/sonotexto"), { "sonotexto
        folder is in place".postln }, { "No sound folder is in your Recordings path, please create
        one with the name 'sonotexto'".postln });
15.
16.      // this executes a document with the SynthDefs required to work.
17.      thisProcess.interpreter.executeFile((Platform.userExtensionDir ++
        "/sonotexto/sonotexto-synths.scd").standardizePath);
```

```
18.      ^"SonoTexto Active"
19.     }
20.
21.     // the class method *rec allows to rec one, or all, the available Buffers when the argument
        is true
22.     // it is possible to warn about which synthdefs are available
23.     *rec { |b1 = 0, b2 = 0, b3 = 0, b4 = 0|
24.      if (b1.asBoolean, { Synth(\b1rec); "Recording b1 mono file".postln }, { "nil" });
25.      if (b2.asBoolean, { Synth(\b2rec); "Recording b2 stereo file".postln }, { "nil" });
26.      if (b3.asBoolean, { Synth(\b3rec); "Recording b3 mono file".postln }, { "nil" });
27.      if (b4.asBoolean, { Synth(\b4rec); "Recording b4 stereo file".postln }, { "nil" });
28.      ^"SonoTexto Rec"
29.     }
30.
31.     // the class method *write allows writing the temporary buffer sounds in the folder
        sonotexto placed in the Recordings Directory
32.     *write { |b1 = 0, b2 = 0, b3 = 0, b4 = 0|
33.
34.      // this fork writes each Buffer in the hard disk when the argument is true. It waits for 60
        milliseconds to avoid writing the same name to all Buffers.
35.      fork {
36.           if (b1.asBoolean, { ~buf1.write(Platform.recordingsDir +/+ "/sonotexto/" ++
        PathName.new("~/.local/share/SuperCollider/Recordings/sonotexto").files.size.asStringT
```

```
oBase(10, 4) ++ "-b1-" ++ Date.localtime.stamp ++ ".wav", "WAV", "int16"); "Writing b1 mono file".postln }, { "nil" });
        0.06.wait;
        if (b2.asBoolean, { ~buf2.write(Platform.recordingsDir +/+ "/sonotexto/" ++ PathName.new("~/.local/share/SuperCollider/Recordings/sonotexto").files.size.asStringToBase(10, 4) ++ "-b2-" ++ Date.localtime.stamp ++ ".wav", "WAV", "int16"); "Writing b2 stereo file".postln }, { "nil" });
        0.06.wait;
        if (b3.asBoolean, { ~buf3.write(Platform.recordingsDir +/+ "/sonotexto/" ++ PathName.new("~/.local/share/SuperCollider/Recordings/sonotexto").files.size.asStringToBase(10, 4) ++ "-b3-" ++ Date.localtime.stamp ++ ".wav", "WAV", "int16"); "Writing b3 mono file".postln }, { "nil" });
        0.06.wait;
        if (b4.asBoolean, { ~buf4.write(Platform.recordingsDir +/+ "/sonotexto/" ++ PathName.new("~/.local/share/SuperCollider/Recordings/sonotexto").files.size.asStringToBase(10, 4) ++ "-b4-" ++ Date.localtime.stamp ++ ".wav", "WAV", "int16"); "Writing b4 stereo file".postln }, { "nil" });
        0.06.wait;
    }
    ^"SonoTexto Write"
}

// the class method *read accesses sonotexto sound folder. A more accurate way is using the class SampleTexto
*read { |server|
```

```
50.      st = Dictionary.new;
51.      st.add(\st -> PathName(Platform.recordingsDir +/+ "/sonotexto/").entries.collect({ arg
         grabacion; Buffer.read(server ? Server.default, grabacion.fullPath) }));
52.      ^"SonoTexto Sounds "
53.     }
54.
55.     // the class method *info says the number of sounds available in the sonotexto folder
56.     *info {
57.      ("SonoTexto has " ++
         PathName.new("~/.local/share/SuperCollider/Recordings/sonotexto").files.size ++ "
         sounds recorded").postcln;
58.     }
59. }
```

**Código fuente del archivo sonotexto-synths.scd**

```
1. // SonoTexto SynthDefs and Buffers
2.
3. (
4. // Assign variables for the Buffers length in seconds
5. ~durb1 = 5;
6. ~durb2 = 5;
7. ~durb3 = 10;
```

```
8. ~durb4 = 10;
9.
10. // Allocate Buffers, odd mono and even stereo
11. ~buf1 = Buffer.alloc(s, s.sampleRate * ~durb1, 1);
12. ~buf2 = Buffer.alloc(s, s.sampleRate * ~durb2, 2);
13. ~buf3 = Buffer.alloc(s, s.sampleRate * ~durb3, 1);
14. ~buf4 = Buffer.alloc(s, s.sampleRate * ~durb4, 2);
15.
16. // Record mono Buffer 1 with 5 seconds length
17. (
18. SynthDef(\b1rec, { |in = 0, ovdub = 0|
19.     RecordBuf.ar(SoundIn.ar(in), ~buf1.bufnum, 0, 1, ovdub, loop:0, doneAction:2);
20. }).add
21. );
22.
23. // Record stereo Buffer 2 with 5 seconds length
24. (
25. SynthDef(\b2rec, { |in1 = 0, in2 = 1, ovdub = 0|
26.     RecordBuf.ar(SoundIn.ar([in1, in2]), ~buf2.bufnum, 0, 1, ovdub, loop:0, doneAction:2);
27. }).add
28. );
29.
```

```
30. // Record mono Buffer 3 with 10 seconds length
31. (
32. SynthDef(\b3rec, { |in = 0, ovdub = 0|
33.     RecordBuf.ar(SoundIn.ar(in), ~buf3.bufnum, 0, 1, ovdub, loop:0, doneAction:2);
34. }).add
35. );
36.
37. // Record stereo Buffer 4 with 10 seconds length
38. (
39. SynthDef(\b4rec, { |in1 = 0, in2 = 1, ovdub = 0|
40.     RecordBuf.ar(SoundIn.ar([in1, in2]), ~buf4.bufnum, 0, 1, ovdub, loop:0, doneAction:2);
41. }).add
42. );
43.
44. // Play Synth Buffer1 Mono. SynthDef has a low cut frequency at 20 Hz.
45. (
46. SynthDef(\b1, { |rate = 1, tg = 0, st = 0, lp = 0, pb1 = 0, ab1 = 1, atb1 = 0.1, sb1 = 4.8, rb1 =
        0.1, ob1 = 0|
47.     var sen, hpf, pan, env;
48.     var trg = Impulse.kr(tg);
49.     sen = PlayBuf.ar(~buf1.numChannels, ~buf1.bufnum, BufRateScale.kr(~buf1.bufnum) *
        rate, trg, st, lp);
50.     hpf = HPF.ar(sen, 20);
```

```
51.     pan = Pan2.ar(hpf, pb1, ab1);
52.     env = EnvGen.kr(Env.new([0, 1, 1, 0], [atb1, sb1, rb1] ), doneAction: 2);
53.     Out.ar(ob1, pan * env);
54. }).add
55. );
56.
57. // Play Synth Buffer2 Stereo. SynthDef has a low cut frequency at 20 Hz.
58. (
59. SynthDef(\b2, { |rate = 1, tg = 0, st = 0, lp = 0, ab2 = 1, atb2 = 0.1, sb2 = 4.8, rb2 = 0.1, ob2
        = 0|
60.     var sen, hpf, env;
61.     var trg = Impulse.kr(tg);
62.     sen = PlayBuf.ar(~buf2.numChannels, ~buf2.bufnum, BufRateScale.kr(~buf2.bufnum) *
        rate, trg, st, lp);
63.     hpf = HPF.ar(sen, 20);
64.     env = EnvGen.kr(Env.new([0, 1, 1, 0], [atb2, sb2, rb2] ), doneAction: 2);
65.     Out.ar(ob2, (hpf * env) * ab2);
66. }).add
67. );
68.
69. // Play Synth Buffer3 Mono. SynthDef has a low cut frequency at 20 Hz.
70. (
```

```
71. SynthDef(\b3, { |rate = 1, tg = 0, st = 0, lp = 0, pb3 = 0, ab3 = 1, atb3 = 0.1, sb3 = 9.8, rb3 =
        0.1, ob3 = 0|
72.     var sen, hpf, pan, env;
73.     var trg = Impulse.kr(tg);
74.     sen = PlayBuf.ar(~buf3.numChannels, ~buf3.bufnum, BufRateScale.kr(~buf3.bufnum) *
        rate, trg, st, lp);
75.     hpf = HPF.ar(sen, 20);
76.     pan = Pan2.ar(hpf, pb3, ab3);
77.     env = EnvGen.kr(Env.new([0, 1, 1, 0], [atb3, sb3, rb3] ), doneAction: 2);
78.     Out.ar(ob3, pan * env);
79. }).add
80. );
81.
82. // Play Synth Buffer4 Stereo. SynthDef has a low cut frequency at 20 Hz.
83. (
84. SynthDef(\b4, { |rate = 1, tg = 0, st = 0, lp = 0, ab4 = 1, atb4 = 0.1, sb4 = 9.8, rb4 = 0.1, ob4
        = 0|
85.     var sen, hpf, env;
86.     var trg = Impulse.kr(tg);
87.     sen = PlayBuf.ar(~buf4.numChannels, ~buf4.bufnum, BufRateScale.kr(~buf4.bufnum) *
        rate, trg, st, lp);
88.     hpf = HPF.ar(sen, 20);
89.     env = EnvGen.kr(Env.new([0, 1, 1, 0], [atb4, sb4, rb4] ), doneAction: 2);
```

```
90.     Out.ar(ob4, (hpf * env) * ab4);

91. }).add

92. );

93.

94. );
```

**Notas**

La numeración de estas notas corresponden al archivo sonotexto.sc.

10: Esta línea informa sí existe la carpeta <sonotexto> en la ruta *Recordings*. La carpeta <sonotexto> debe crearse manualmente y colocarse en la ruta indicada.

13: Esta línea llama al archivo sonotexto-synths.scd.

19: El número 1 corresponde al valor booleano *true* y el 0 a *false*. Estos valores son convertidos a valores booleanos en la línea 20 con el mensaje `.asBoolean`.

41: El método de clase `*read` permite leer los archivos guardados en el disco duro. Sin embargo, requiere una programación compleja. La lectura de sonidos la implementé en la clase SampleTexto.

## A.2 SampleTexto

**Archivo:** sampletexto.sc

**Lenguaje de programación:** sclang SuperCollider

**Desarrollado:** 2020 – 2021

**Autor:** Hernani Villaseñor

**Plataforma:** GNU/Linux, Windows, Mac OS, Raspberry Pi OS

**Repositorio:** https://github.com/hvillase/sampletexto

```
1. // SampleTexto
2. // A class to create synths form a sound files in a folder
3.
4. SampleTexto {
5.
6.      var <sampleDictionary;
7.      var <sampleMonoDictionary;
8.      var <sampleStereoDictionary;
9.
10.     // Constructor (Método constructor)
11.     *new {
12.      ^super.new.init;
13.     }
14.
```

```
15.  // Read folder of Recordings path (lee carpetas del directorio Recordings)
16.  init { |server, path = "/sampletexto/"|
17.
18.   sampleDictionary = Dictionary.new;
19.
20.   sampleDictionary.add(\smp -> PathName(Platform.recordingsDir +/+
      path).entries.collect({ |grabacion| Buffer.read(server ? Server.default,
      grabacion.fullPath) }));
21.  }
22.
23.  // Create a mono Dictionary and a stereo Dictionary (crea un diccionario mono y uno
      estéreo)
24.  stsel {
25.   sampleMonoDictionary = sampleDictionary[\smp].select{ |item| item.numChannels ==
      1 };
26.   sampleStereoDictionary = sampleDictionary[\smp].select{ |item| item.numChannels ==
      2 };
27.  }
28.
29.  // Read sounds from the general Dictionary (lee sonidos de la carpeta general)
30.  st { |num1 = 0|
31.   ^this.sampleDictionary[\smp][num1];
32.  }
33.
```

```
34.  // Read sounds from mono Dictionary (lee sonidos de la carpeta mono)
35.  stm { |num2 = 0|
36.   ^this.sampleMonoDictionary[num2];
37.  }
38.
39.  // Read sounds from stereo Dictionary (lee sonidos de la carpeta estéreo)
40.  sts { |num3 = 0|
41.   ^this.sampleStereoDictionary[num3];
42.  }
43.
44.  // Create mono synths from mono Dictionary (crear Synths del diccionario mono)
45.  monosynth {|num = (sampleMonoDictionary.size - 1)|
46.       (0..num).do{|it|
47.            this.sampleMonoDictionary[it].normalize;
48.            SynthDef("m%".format(it), {|rate=1, sp=0, at=0.001, sus=1, rel=0.001, pan=0,
     amp=1, out=0|
49.                 var son, hpf, pne, env;
50.                 son=PlayBuf.ar(1, this.sampleMonoDictionary[it].bufnum, rate, 1, sp *
     this.sampleDictionary[\smp][it].numFrames, 0);
51.                 hpf=HPF.ar(son, 20);
52.                 pne=Pan2.ar(hpf, pan, amp);
53.                 env=EnvGen.kr(Env.new([0, 1, 1, 0], [at, sus *
     this.sampleMonoDictionary[it].duration, rel]), doneAction: 2);
```

```
54.                     Out.ar(out, pne * env);
55.                 }).add;
56.             }
57.     }
58.
59.     // Create stereo synths from stereo Dictionary (crear Synths del diccionario estéreo)
60.     stereosynth {|num = (sampleStereoDictionary.size - 1)|
61.             (0..num).do{|it|
62.                 this.sampleStereoDictionary[it].normalize;
63.                 SynthDef("s%".format(it), {|rate=1, sp=0, at=0.001, sus=1, rel=0.001, amp=1,
        out=0|
64.                     var son, hpf, env;
65.                 son=PlayBuf.ar(2, this.sampleStereoDictionary[it].bufnum, rate, 1, sp *
        this.sampleStereoDictionary[it].numFrames, 0);
66.                     hpf=HPF.ar(son, 20);
67.                     env=EnvGen.kr(Env([0, 1, 1, 0], [at, sus *
        this.sampleStereoDictionary[it].duration, rel]), doneAction: 2);
68.                     Out.ar(out, (son * env) * amp);
69.                 }).add;
70.             }
71.     }
72.
73.     // Specify the number of samples in a folder (cantidad de sonidos en la carpeta)
```

```
74.     info {
75.       ("total: " ++ sampleDictionary[\smp].size).postln;
76.       ("mono: " ++ sampleMonoDictionary.size).postln;
77.       ("stereo: " ++ sampleStereoDictionary.size).postln;
78.     }
79. }
```

## A.3 Ptexto

**Archivo:** ptexto.sc

**Lenguaje de programación:** sclang SuperCollider

**Desarrollado:** 2021

**Autor:** Hernani Villaseñor

**Plataforma:** GNU/Linux, Windows, Mac OS, Raspberry Pi OS

**Repositorio:** https://github.com/hvillase/ptexto

```
1. // Sequential pattern for modulo operator
2.
3. Ptexto : Pattern {
4.
5.   // list: the array of values, modVal: modulo value, sumVal: adds to modulo
6.   var <>list, <>modVal, <>sumVal, <>repeats;
```

```
7.
8.    // Constructor
9.    *new { |list, modVal = 1, sumVal = 0, repeats = 1|
10.         ^super.newCopyArgs(list, modVal, sumVal, repeats)
11.   }
12.
13.   // Embed the values of the list into the stream
14.   embedInStream { |inval|
15.         var listVal = list;
16.         var item;
17.
18.         repeats.do({ |i|
19.   // Apply the modulo operation to the list values
20.               item = (listVal[i % list.size].mod(modVal) + sumVal);
21.   // Allows to place the resulting values from Ptexto computation in the pattern stream
22.               inval = item.embedInStream(inval);
23.         });
24.         ^inval;
25.   }
26. }
```

# B. Instalación de las clases SonoTexto

Primero hay que ingresar a los siguientes repositorios y descargar o clonar el contenido de cada clase:

https://github.com/hvillase/sonotexto

https://github.com/hvillase/sampletexto

https://github.com/hvillase/ptexto

Las clases se descargan en un formato comprimido ZIP desde la pestaña *code*. Si se opta por esta opción el siguiente paso sería descomprimir el contenido. También se puede clonar el repositorio desde la terminal con el comando:

$ git clone https://github.com/hvillase/sonotexto

Luego hay que colocar las carpetas descomprimidas o clonadas en la ruta de SuperCollider llamada *Extensions*. Para saber dónde se encuentra esa carpeta se puede usar el comando `Platform.systemExtensionDir` en el IDE de SuperCollider, el cual nos indica la ruta del directorio de extensiones en la ventana *post*.

En el caso de la clase SonoTexto se requiere crear una carpeta llamada sonotexto en la dirección *Recordings*. Para encontrar la ruta podemos usar el comando `Platform.recordingsDir` en el IDE de SuperCollider.

Una vez que hemos colocado las carpetas en la ruta *Extensions* y generado la carpeta sonotexto en *Recordings* abrimos SuperCollider, iniciamos el servidor y probamos las clases. Para ello nos auxiliamos de los ejemplos incluidos en lo archivos de ayuda de las clases dentro de SuperCollider. Para ver el contenido de estos archivos se pueden utilizar los comandos `SonoTexto.help`, `SampleTexto.help` y `Ptexto.help` en el IDE de SuperCollider.

Si se está familiarizado con el uso de quarks en SuperCollider, una opción es descargar sonotexto-quark, el cual incluye todas las clases. Para ello es necesario ir al siguiente repositorio: https://github.com/hvillase/sonotexto-quark.

Una vez ahí, descargamos o clonamos el quark y lo colocamos en la carpeta *downloaded-quarks*. Esta carpeta se encuentra en el mismo directorio que la carpeta *Extensions*. Abrimos SuperCollider y escribimos el comando `Quarks.gui`. Luego presionamos el símbolo [+] que se encuentra al lado del nombre de la carpeta sonotexto-quark que cambiará a color verde. Por último, presionamos el botón *Recompile class library* que se encuentra en la parte superior derecha. De esta manera queda activado el quark y podemos hacer uso de las tres clases.

Otro método para instalar el quark que acabamos de descargar es declarar el comando `Quarks.install("~/ruta_del_quark_descargado/")` dentro del IDE de SuperCollider. Para ello se requiere sustituir la ruta de descarga de ejemplo que está entre comillas. Asimismo, es posible instalar sonotexto-quark desde el repositorio git, para ello escribimos el siguiente comando en el IDE de SuperCollider:

Quarks.install("https://github.com/hvillase/sonotexto-quark")

# C. Práctica artística

Este anexo contiene las presentaciones y ejemplos realizados con SonoTexto durante el periodo de la investigación. Están divididas en presenciales y en línea. Algunos de ellos están descritos en la sección 4.3. Los videos de las presentaciones están organizados en la siguiente lista de YouTube: https://youtube.com/playlist?list=PLFB-yISQMraUw9YtiBXH6vaEuA38oFYkN. La presentación en línea Transnodal marcada con el número 6 no tiene registro y la presentación 10th Algorave Birthday marcada con el número 7 está en la siguiente dirección de Archive: https://archive.org/details/algorave-10-h3v.

*Presenciales:*

1 *Concierto de live coding*. Galería el Rule, 26 de junio de 2019. Ciudad de México.

2 *Dar forma al espacio*. 1° Festival Expresiones Contemporáneas, 16 de octubre de 2019. Puebla.

3 *Iterar el espacio*. Resonancias XXIX, 7 de noviembre de 2019. FaM UNAM, Ciudad de México.

4 *SonoTexto*. 5° International Conference on Live Coding, 6 de febrero de 2020. Universidad de Limerick, Limerick.

*En línea:*

1 *SonoTexto: soundscape live coding*. Euleroom Equinox, 21 de marzo de 2020. Organizado por Toplap.

2 8:08pm La hora del LIVECODER, 6 de mayo de 2020. Organizado por Toplap Barcelona.

3 Campamento extendido: impendingVoid/, 3 de julio de 2020. Organizado por Posternura Records, Chile.

4 *Paisaje Sonoro Emergente*. XV Coloquio de Alumnos del Programa de Maestría y Doctorado en Música, 27 de noviembre de 2020. Organizado por el Posgrado en Música UNAM.

5 Algorave Brasil, 13 de diciembre de 2020. Organizado por Algorave Brasil.

6 Transnodal, 19 de febrero de 2021. Organizado por Toplap.

7 *nTexto*. 10th Algorave Birthday, 19 de marzo de 2022. Organizado por Algorave.

## D. Portafolio de la tesis

El portafolio de la tesis puede ser consultado en el siguiente sitio web:

https://hernanivillasenor.com/html/sonotexto.html

# E. Línea temporal de tecnologías para el live coding

Este anexo presenta una línea temporal de algunos lenguajes de programación y librerías utilizadas para hacer live coding con referencia a la publicación del Manifiesto de live coding de Toplap y al programa MUSIC.

| | | |
|---|---|---|
| 1957 | \|- | MUSIC |
| **Utilizadas en el live coding** | -\| | |
| 1970 | \|- | Forth |
| 1986 | \|- | Csound |
| 1987 | \|- | Perl |
| 1996 | \|- | SuperCollider |
| | \|- | Pure Data |
| **Desarrolladas para el live coding** | -\| | |
| 2003 | \|- | ChucK |
| 2004 | \|- | **Manifiesto de live coding de Toplap** |
| 2005 | \|- | Fluxus |
| 2008 | \|- | Conductive |
| 2009 | \|- | TidalCycles |
| | \|- | ixi lang |
| 2011 | \|- | Extempore |
| 2012 | \|- | SonicPi |
| | \|- | Gibber |
| 2014 | \|- | CaosBox |
| 2015 | \|- | Estuary |
| | \|- | FoxDot |
| 2016 | \|- | Mégra |
| 2017 | \|- | INSTRUMENT |
| 2018 | \|- | Hydra |
| 2019 | \|- | Sema |
| | \|- | Flok |
| | \|- | SonoTexto |
| 2020 | \|- | seis8s |
| | \|- | MIRLCa |
| 2022 | \|- | Strudel |