Expression in Live Coding: Gestural Interaction for Machine Musicianship

GEORGIOS DIAPOULIS

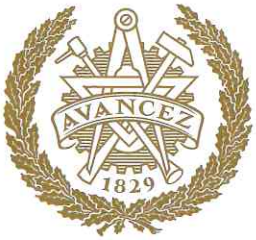
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

This thesis concerns musical live coding, which is a performance practice for musical improvisation with computing systems. Live coding is a neologism that first appeared in 2003 within the computer music community to describe an experimental practice for music-making. Since then, live coding has evolved into a world-wide community of practitioners and academics that spans different disciplines, from music and dance to weaving and programming education.

A typical live coding performance involves one or more performers writing computer programs on stage and sharing their screens with the audience to make the process technically transparent. The main idea is that we share our screens to expose our technical mistakes, and not to use technology as a "safety net" to hide our mistaken beliefs and personal technical limitations.

This thesis contributes technical developments and theoretical studies with a particular focus on human perception and cognition. I am particularly focusing on understanding to what extent we use sub-conscious processes, also known as pre-reflective processes, when interacting with musical systems in the writing of computer programs.



CHALMERS
UNIVERSITY OF TECHNOLOGY

Thesis for The Degree of Doctor of Philosophy

Expression in Live Coding: Gestural Interaction for Machine Musicianship

Georgios Diapoulis

Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden, 2023

Expression in Live Coding: Gestural Interaction for Machine Musicianship

Georgios Diapoulis

ISBN 978-91-7905-926-2
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 5392.
ISSN 0346-718X

Department of Computer Science and Engineering
Division of Interaction Design and Software Engineering
Chalmers University of Technology | University of Gothenburg
SE-412 96 Göteborg,
Sweden
Phone: +46(0)31 772 1000

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2023.

To my parents

Expression in Live Coding: Gestural Interaction for Machine Musicianship

Georgios Diapoulis

Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg

Abstract

This thesis is centered on the performance practice of musical live coding, which can be described as on-the-fly decision-making for computer music performance and blurs the lines between programming languages and computer interfaces. I specifically focus on live coding as a human activity based on serial skilled actions, and I discuss how we can interact gesturally with interfaces that are modified dynamically. I develop a system to address how to live code using continuous gestural interactions, and examine an unusual strategy of information processing which centers on a bottom-up approach. I then present a theoretical account of sensorimotor control in live coding, which examines pre-reflective processes during performance. Two more theoretical contributions address bodily gestures in various performance systems by presenting a conceptual framework, and an additional study presents a conceptual tool for developing agent-based systems in live coding. Observations of performance practices and systems are employed throughout the thesis to examine how different practitioners may use the systems, and what conceptual abstractions can be inferred. The question of how to facilitate creativity is discussed from both a theoretical and a practical perspective, touching on how the resultant newly produced knowledge may be transferred to practitioners. A particular focus is on live performance, visualization, and both human and machine listening.

The thesis findings present (i) visual and conceptual representations of live coding validated by cognitive mechanisms and cognitive paradigms, (ii) extensive reflections on practice and systemic modes of knowing, (iii) technical contributions for building of performance systems and performance structural analysis and (iv) theoretical accounts that contribute to the live coding literature. This work address both human modalities – audition, motor skills – and system modalities – notation, and the presence of software agents. I present a conception of various kinds of *interactivity variations*, a term used to designate various manners of gestural interaction that may arise in performance systems. I argue that musical live coding should incorporate radical experimentation with craft practices, pointing to a future practice wherein risk becomes clearly apparent in our gestural expressions.

Keywords

live coding, live programming, expressive interaction, liveness, music interaction, musical interfaces, music performance, music perception, gestural control, music information retrieval

List of Publications

Appended publications

This thesis is based on the following publications:

[**Paper I**] **G. Diapoulis** and P. Dahlstedt. "An analytical framework for musical live coding systems based on gestural interactions in performance practices." *Proceedings of the International Conference on Live Coding (ICLC).* Valdivia, Chile. (2021).

[**Paper II**] **G. Diapoulis** and P. Dahlstedt. "The creative act of live coding practice in music performance." *Psychology of Programming Interest Group (PPIG), Doctoral Consortium.* York, UK. (2021).

[**Paper III**] **G. Diapoulis**, I. Zannos, K. Tatar and P. Dahlstedt. "Bottom-up live coding: Analysis of continuous interactions towards predicting programming behaviours." *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME).* Auckland, New Zealand. (2022).

[**Paper IV**] **G. Diapoulis**. "Livecode me: Live coding practice and multimodal experience." *Proceedings of the 33rd Annual Workshop of the Psychology of Programming Interest Group (PPIG).* London, UK. (2022).

[**Paper V**] **G. Diapoulis** and M. Carlé. "Reproducible musical analysis of live coding performances using information retrieval: A case study on the Algorave 10th anniversary." *Proceedings of the International Conference on Live Coding (ICLC).* Utrecht, Netherlands. (2023).

[**Paper VI**] **G. Diapoulis**. "Liveness and machine listening in musical live coding: A conceptual framework for designing agent-based systems." *Proceedings of the 4th Conference on AI Music Creativity (AIMC).* Brighton, UK. (2023).

[**Paper VII**] **G. Diapoulis**. "Musical live coding in relation to interactivity variations." *Organised Sound 28.2.* Cambridge University Press, UK. (2023).

Other publications

The following publications were published before or during my PhD studies. However, they are not appended to this thesis because they were published before my PhD studies [a-d], unrelated to the thesis, or overlapping with the thesis content. An updated list of all my publications is available on my website[1], ORCID[2] and Google Scholar[3].

[**a**] **G. Diapoulis** and I. Zannos. "A minimal interface for live hardware coding." *Live interfaces*. York, UK. (2012).

[**b**] **G. Diapoulis** and I. Zannos. "Tangibility and low-level live coding." *International Computer Music Conference*. Athens, Greece. (2014).

[**c**] **G. Diapoulis** and M. Thompson. "Kinematics feature selection of expressive intentions in dyadic violin performance." *10th International Conference of Students of Systematic Musicology (SysMus17)*. London, UK. (2017).

[**d**] M. R. Thompson, **G. Diapoulis**, T. Himberg, and P. Toiviainen. "Interpersonal coordination in dyadic performance." *Routledge Companion to Embodied Music Interaction*. (eds.) LeSaffre, Leman & Maes, Routledge. (2017).

[**e**] **G. Diapoulis**, C. Rosas, K. Larsson and W. Kropp. "Person identification from walking sound on wooden floor." *Proceedings of the Euronoise*. Heraklion, Greece. (2018).

[**f**] B. Briere de la Hosseraye, **G. Diapoulis**, F. Egner and H. Wang. "A case study on workstation dependent acoustic characterization of open plan offices." *EU Horizon report for the ACOUTECT project (grant agreement 721536)*. (2021).

[**g**] **G. Diapoulis**. "Primary and secondary aspects of musical gestures in live coding performance." *Proceedings of the 14th International Conference of Students of Systematic Musicology (SysMus21)*. Aarhus, Denmark. (2021).

[**h**] V. Agiomyrgianakis, I. Svoronos-Kanavas, I. Zannos and **G. Diapoulis**. "Synerg(e)ia: A networked collaborative live coding environment." *Proceedings of the International Conference on Live Coding (ICLC)*. Valdivia, Chile. (2021).

[**i**] K. Tatar, P. Ericson, K. Cotton, P. T. N. del Prado, R. Batlle-Roca, B. Cabrero-Daniel, S. Ljungblad, **G. Diapoulis** and J. Hussain. "A shift in artistic practices through artificial intelligence." *Submitted to Leonardo. arXiv preprint arXiv:2306.10054*. (2023).

[1] Georgios Diapoulis personal webpage, `http://diapoulis.gitlab.io`
[2] `https://orcid.org/0000-0002-3101-1875`
[3] `https://scholar.google.com/citations?user=Lh4Oz9EAAAAJ`

Other dissemination (selection)

The following events are part of the research dissemination I did during the second half of my doctoral studies. These are either musical performances or public presentations in smaller research venues or for non-expert audiences.

[**1**] **G. Diapoulis**. "Perceptual and technical aspects of live coding music performance using statistical learning." *Presentation at DCAC 2021 – Digital Culture & AudioVisual Challenges, Cyberspace.* Corfu, Greece. May, 2021.

[**2**] **G. Diapoulis**. "Sonifying the collected rewritings of Click Nilson." *Live performance at ICLC2021, Cyberspace.* Valvidia, Chile. December, 2021.

[**3**] **G. Diapoulis**. "Musical live coding performance (dyadic agents)." *Lindblad Electronic Music Meet, Artisten.* Gothenburg, Sweden. April, 2022.

[**4**] **G. Diapoulis**. "Musical live coding in relation to expressive interaction." *Presentation at Nordic Summer University (NSU), summer school.* Oslo, Norway. August, 2022.

[**5**] **G. Diapoulis**. "Continuous gestural interactions in live coding." *Presentation at the Workshop Embodied Perspectives on Musical AI (EmAI), U. of Oslo.* Oslo, Norway. November, 2022.

[**6**] S. Kalonaris, I. Zannos and **G. Diapoulis**. "Duel Revisited." *Networked live coding performance at XNPM22 Xenakis Networked Performance Marathon, Athens Conservatoire.* Athens, Greece. December, 2022.

[**7**] **G. Diapoulis**. "From 'no-code' to 'slow-code': A minimalistic approach to musical live coding." *Presentation, Nordic Summer University (NSU) winter school, Au JUS artist run space.* Brussels, Belgium. March, 2023.

[**8**] **G. Diapoulis** and Viktor Kudryashov. "Collaborative live coding performance with music and visualizations." *Nordic Summer University (NSU) winter school, Au JUS artist run space.* Brussels, Belgium. March, 2023.

[**9**] G. Strautmane and **G. Diapoulis**. "Codifications of Musical Code." *Exhibition at Cesvaine Palace.* Cesvaine, Latvia. July, 2023.

Acknowledgment

I particularly want to thank Palle Dahlstedt for the supervision. His influence on this work and my journey as a doctoral student is hard to express in words. I genuinely thank you! I also thank Staffan Björk for his kindness in being my examiner; his minimalistic pieces of advice impact the work presented here, either directly or indirectly. I am also grateful to my second and third supervisors, Mohammad Obaid and Kivanç Tatar, for their support and advice. Mohammad, I also thank you for being an excellent manager. Kivanç, thank you for all the reviews and expert feedback. I particularly thank my former supervisor from the Division of Applied Acoustics Jens Forssén, and my former examiner, Jens Ahrens, for facilitating my transfer to Interaction Design. I am also thankful to Monica Billger, for helping me with the move between the departments.

First, I would like to thank Mohammad, Morten, Josef and Jasmina, who trusted me to teach during my 2.5 years. I also thank Thommy for his kindness and unconditional willingness to help with everything related to the IxD studio and electronic equipment. I am grateful to Jasmina Maric for the unique experience of the Creative Coding course for adolescent girls. A special thanks to Kelsey for co-teaching with me, and all people involved, Lekshmi, Pauline, Natasha, Razan and Christie.

Special thanks go to Paweł Woźniak and Sara Ljundblad, as they both influence my academic work – regardless of our minimal academic interactions. I thank you both. Particularly Paweł for his sharpness and capacity to advise me in less than a 5-minute discussion on how to present my work (which had an influence on the very title of this manuscript). Special thanks go to all fellow doctoral students of IxD, Mafalda, Sjoerd, Kelsey, Ziming, and Yuchong. Mafalda, I thank you for your reviews and to-the-point feedback. I am also grateful to have such beloved officemates Linda, Joel, Handy, Razan, Peter, Ranim and Ann-Sophie. A more special thanks to all administrators and staff, especially to IxD people Cecilia, Lotta and Lasse, all colleagues on the 3rd floor of Kuggen, and the GU staff for their kind presence.

Needless to say, without the PhD support of Chalmers/GU it would have been impossible to complete this work. I warmly thank the head of PhD education Agneta Nilsson and all study directors of CSE. Also, the PhD administrators and particularly Clara Oders for making my life easier.

A special and genuine thank I own to Moyra. Without your support this

journey would have been incomplete.

During my studies in acoustics, I traveled as part of the ACOUTECT research network. I thank all the people in Eindhoven for hosting me, especially Maarten Hornikx. I also thank Jacques Cuenca for hosting my secondments in Leuven. I am grateful to Antonio, Alessia, Augusto, Julie and Felix for our time together somewhere in Europe.

I am also thankful to the Nordic Summer University (NSU) people, especially Circle 2, for their support and trust. I feel lucky I have had the opportunity to attend such a unique nomadic organization, and I genuinely thank you for our discussions, laborious discord, and fun time somewhere in the Nordic/Baltic region. The opportunity to get exposed to experts in STS, law and other research fields is a unique opportunity for me.

I want to thank my former supervisors during my master's studies in Finland, Marc Thompson, Pasi Saari and Petri Toivianen. I am truly grateful for the education you offered and feel lucky to have had this opportunity. Marc, I warmly thank you for the opportunities you unconditionally offered me. Most importantly, thank you for your kindness in supporting and advising me when I needed it! Your influence made a change.

Furthermore, I want to thank Nick Collins for his kindness in advice me on postgraduate studies and his suggestion to attend the masters program Music, Mind and Technology in the U. of Jyväskylä. I am also grateful for the interest he showed in my early works. I am also grateful to Dan Stowell for his kind, invisible support via the SC3 mailing list and his trust in accepting my work at the SuperCollider Symposium in London. Given our minimal interactions, it is truly remarkable how much influence Nick and Dan have had on my academic development. I warmly thank you both.

A special thanks goes to Ioannis Zannos for his continuous support throughout the years. It is out of the question that without his support, I would not have been able to start a master's program, not to say a PhD. Ianni, I genuinely thank you for all the discussions, writings, journeys, teachings, and performances I participated in. I also thank Martin Carlé for all our philosophical discussions, academic collaborations, and rigorous discourse.

Last but not least, I want to thank my Greek academic friends Symeon Delikaris-Manias, Vassilis Papadourakis, Xenophon Zabulis and Danae Theodosopoulou for their support during my studies. I thank you for all this time on some video call. I also thank my Greek acoustician friends Georgios Zachos and Nikos Roubakis. I also thank all my Göteborgsvänner, Athenian and Cretan friends for their support. My love goes to my family, especially my parents and my sister, for their unconditional support.

To conclude, I am truly grateful to Tim Perkis for proofreading this manuscript. No words can express my gratitude, I warmly thank you. I also thank Babak Ahteshamipour for taking care of the magnificent front cover artwork.

Göteborg
September 10, 2023

Foreword

This thesis corresponds to the second half of my doctoral studies at Chalmers University of Technology and University of Gothenburg. I did 50% of my doctoral studies in Applied Acoustics, Department of Architecture and Civil Engineering as part of the EU project ACOUTECT (Marie Sklodowska-Curie grant agreement number 721536). The following thesis shows my work in the second half of my studies, since May 2021, at the Division of Interaction Design and Software Engineering, Unit of Interaction Design, Department of Computer Science and Engineering.

Abbreviations

- **ACM**: Association for Computing Machinery
- **AI**: Artificial Intelligence
- **AR**: Augmented Reality
- **CHI**: ACM Conference on Human Factors in Computing Systems
- **CPU**: Central Processing Unit
- **DM**: Direct Manipulation
- **DMI**: Digital Musical Instrument
- **GUI**: Graphical User Interface
- **HCI**: Human-Computer Interaction
- **ICLC**: International Conference on Live Coding
- **ICMC**: International Computer Music Conference
- **IDE**: Integrated Development Environment
- **ISMIR**: International Society for Music Information Retrieval
- **LLM**: Large Language Model
- **MIR**: Music Information Retrieval
- **NIME**: International Conference on New Interfaces for Musical Expression
- **PD**: Pure Data visual programming language
- **SC3**: SuperCollider programming environment
- **SE**: Software Engineering
- **SMC**: Sound and Music Computing Conference
- **TEI**: ACM Conference on Tangible, Embedded and Embodied Interaction

Contents

Chapter 1

Introduction

Everyone has daily exposure to numerous hours in front of a computer or a mobile phone. However, we do not know how to interact with computers in a "human-friendly" manner. Computers have been designed based on the principles of logic, so every step we take in an algorithmic process has to "make sense" for the computer language and interface. Consequently, users can be unclear about how to interact without putting in logical and conscious effort.

This thesis investigates how users can interact with generative algorithms in an intuitive manner without making conscious effort while using creative applications, particularly for music-making. The focus of the work presented here is on live coding, which blurs the line between programming languages and computer interfaces. This manuscript is primarily concerned with an embodied perspective on music psychology and perception, exploring how humans who are engaged with these cybernetic systems express themselves through musical code. The main interest is pre-reflective processes during a live coding performance; that is, sub-conscious or fast processes. This work also presents practical implementations and conceptual frameworks of such interactive systems, focusing on both the human physical movements and the systemic characteristics of the machine. Finally, an approach to live coding is discussed, exemplifying how one may write bottom-up programs by progressively constructing programmable levels of abstraction. This bottom-up approach presents unique characteristics that can seamlessly integrate the human and machine aspects of the cybernetic system, creating a significant shift that enables pre-reflective processes during live coding.

What makes music so different? Practical experience suggests that music offers an ideal ground for conducting research where mind and body are coordinated in a way similar to the meditative states found in eastern philosophical thought. This is a unique opportunity for phenomenological studies, that is, the philosophy of human experience and philosophy as theoretical reflection.

The work presented here is influenced by generative arts, systems and process art, where the artist is engaged in a bricolage of modern technologies. A particular focus is on minimalistic expressions of art and programming constructs. I start with a brief introduction to interactive music systems and

proceed to musical live coding. Following this, I move to the specifics of the thesis by introducing the contributions and research questions, and by explaining how to read this manuscript.

1.1 Machine musicianship

Machine musicianship combines algorithmic music and human cognition [1]. The main quest is to develop novel software and hardware for making *interactive music systems* [2] to be used for musical performance. Interactive music systems are designed to recognize and generate music that plausibly meets our cognitive abilities, sensory capabilities, and musical preferences and aesthetics. There is a fine line between what we perceive as musical and what we perceive as random sounds, and this varies across individuals and also depends on levels of musicianship. Musicians using interactive music systems have to decode and adapt to the meta-instruments they develop for real-time performance and improvisation, and this certainly is not something that has been taught during their traditional musical training.

This thesis centers on an embodied perspective for machine musicianship, not using embodied agents as in robotics, but on how theories of embodied cognition may inform the systems we develop. Thus, there is herein a distinct focus on motor theories of perception. Musicians are known to be experts in sensorimotor control, which makes the task of crafting and developing software and hardware interfaces a significant challenge. Today there is certainly no problem providing low enough latency in response times between the "sensing" and "acting" of computer programs. Even though this has been the case for the last two decades or so, the computational load may still be challenging, even when heavy computational resources are available. Expert musicians can perceive any unwanted time delays more than 5-10ms, the lower threshold of a typical sound card operating at 48kHz with 512 samples buffer size.

The main technique for developing software that can co-perform with humans in real-time involves "divide-and-conquer": that is, the development of specialized components for different tasks. For instance, when sensing the acoustic environment, a signal is captured using a microphone and read by the computer, and specialized software extracts various musical characteristics. Pitch estimation, envelope following, beat-tracking, and music segmentation are some of the important steps needing to be carried out during machine listening. Likewise, various strategies can be followed in music generation to generate musical material with diverse qualities. All these components can act autonomously using software agents on different levels to handle temporal, structural, harmonic and timbral parameters.

This characteristic *machine autonomy* means that musicians must acquire new skills as we augment human musicianship with meta-composition tools. Traditional musical instruments are largely agnostic with regards to autonomy, as there will be no sound produced when there is no human action. The question of agency arises in electronic and computer music where there are autonomous systems involved, as it becomes non-obvious which entity is acting

to produce a particular musical outcome, the machine or a human performer?

1.2 New interfaces for musical expression

The new millennium started with emerging forms of musical expression. A new spin-off conference of the ACM Conference on *Human Factors in Computing Systems (CHI)*, called *New Interfaces for Musical Expression*, or *NIME*, began in 2001 from a small CHI workshop. It is an annually recurring and established venue producing high-quality research in what we can broadly call music-making with computers. The focus is on interfaces and instruments that generate music, and the abbreviation NIME can have multiple meanings. In 2017, a NIME Reader, [3] was published with a selected collection of articles from 2001 to 2015. This collection of articles demonstrates the research output of the community, which has contributions on sensors [4]–[7], communication protocols [8], [9], theoretical studies including instrument and interface designs [10]–[14] and new musical interfaces and manners for musical expression[15]–[21].

Various seminal articles have been published at NIME with significant influence in music technology in general, such as the OSC protocol [8], on-the-fly programming [18], the reacTable [16], the use of the web browser as a synthesizer [21] and more. Indicatively, the OSC protocol has become a standard for sending and receiving sound data, and is used in most advanced music-making and multimedia software, for instance, when making 3D graphics and wanting to synchronize the sound with the moving images. On-the-fly programming is an equivalent term that corresponds to live coding. The reacTable is perhaps the first musical interface that has reached a broader audience, most notably used by the famous Icelandic singer-songwriter Björk[1] in her performances, and the web browser today is getting more and more attention as a platform for real-time audio. Indicatively, CSound, which has long been one of the most influential platforms for real-time sound synthesis, now has a web IDE[2].

1.3 Musical live coding

Live coding [22], also known as on-the-fly programming [18], or just-in-time programming [23], is a form of interactive or conversational programming [24] wherein the user converses "with the computer in its own native language" [25, p. 2]. From a technical standpoint, the precursors of live coding are similar to *hot-swapping* practices used in hardware, which is the case where a modular hardware component is "hot-plugged" into a system without shutting down the running system.

From a theoretical perspective, live coding is a situation where practitioners are "thinking-in-action" [26]. Everyday actions like grasping a glass of water do not require conscious awareness. However, live coding is known to be a highly demanding cognitive task where automatic and fast processes are

[1] `https://en.wikipedia.org/wiki/Reactable` (accessed 2023-08-23)
[2] `https://ide.csound.com/`

perhaps not in the picture. In live coding, Emma Cocker [26, p. 111] discussed this "dual principle of slowness and speed." She seized the opportunity to discuss the *kairotic*[3] dimension of live coding, exemplifying the timely nature of decision-making during a performance.

In everyday life, the live coder confronts the activity of coding in a radically different manner than a software engineer. Live coding is *programming for fun* [27], a "non-productive" approach that does not emphasize innovation and robustness [28]. Failure is engraved in the live coding culture, exemplified in the exploratory nature of the logical processes during a performance, where instead of the `if/then/else` procedural logic, a live coder speculates *what if* I run this code chunk. All in all, in live coding, "code is a process to be experienced" [25, p.217].

1.4 Research approach

1.4.1 Personal statement/background

The work presented here is highly interdisciplinary. It builds upon my formal training as a music psychologist, alongside my background in materials science, acoustics and computer science. As a music practitioner, I have been involved in algorithmic composition and interface development since 2008, and as a hobbyist computer music enthusiast since 1999.

I have been involved in academic teaching for more than 10 years, conducting artistic installations and musical performances, participating and organising expert user focus groups, cultural events, academic seminars, and collaborating in interdisciplinary research projects. As a student, employee or visitor/intern, I have worked in five different European countries across seven different academic institutions. My undergraduate studies began in the school of pure sciences, then I moved to humanities and, finally, to engineering.

1.4.2 Research approach and methods

The research approach is a blend of quantitative and, mostly, qualitative methodology, using a broad spectrum of research methods and research designs from information retrieval and digital design to observations and theory-making based on ethnography and embodied cognition. I claim my worldview to be a pragmatic approach with influences from constructivism. Systems thinking is a large influence on this thesis. Thus, I do not intend to conduct measurements and evaluations related to live coding, but to find emerging patterns by mapping relationships between things [29]. Empirical evidence and subjective percepts are central to my work, ranging from the theoretical frameworks I rely upon, such as embodied music cognition [30], [31] and other phenomenological perspectives on science [32], [33]. Finally, I have carried out meta-cognitive learning processes, such as reflective diaries and systematic documentation of

[3] From the greek word *kairós*, which would be literally translated as weather, but has multiple meanings related to time, period and opportunity. `https://en.wiktionary.org/wiki/%CE%BA%CE%B1%CE%B9%CF%81%CF%8C%CF%82` (accessed 2023-08-24).

the related literature by means of an extensive annotated bibliography. By using a blend of induction, deduction, and abduction, I form theories on live coding by conducting observations and I provide guidelines for making informed decisions when designing, practicing, and studying live coding systems.

1.5 Contributions

In this thesis, I present four studies to support my contributions and strengthen my work and the work of others. The contributions of this thesis are clustered as *conceptual*, *methodological*, and *applied* contributions to support the reader's ability to navigate and comprehend this thesis, and is inspired by Sara Ljungblad's doctoral dissertation [34]. I have assigned one particular category to each article, although, as is shown on the Venn diagram, these categories overlap in some instances. The purpose of this mental map is to familiarize the reader with the literature and provide a high-level understanding of the different territories explored in this thesis (Figure 1.1). In the section 1.8 "Structure of the thesis", I present a table of the studies, where I show the influence of each article on each study.

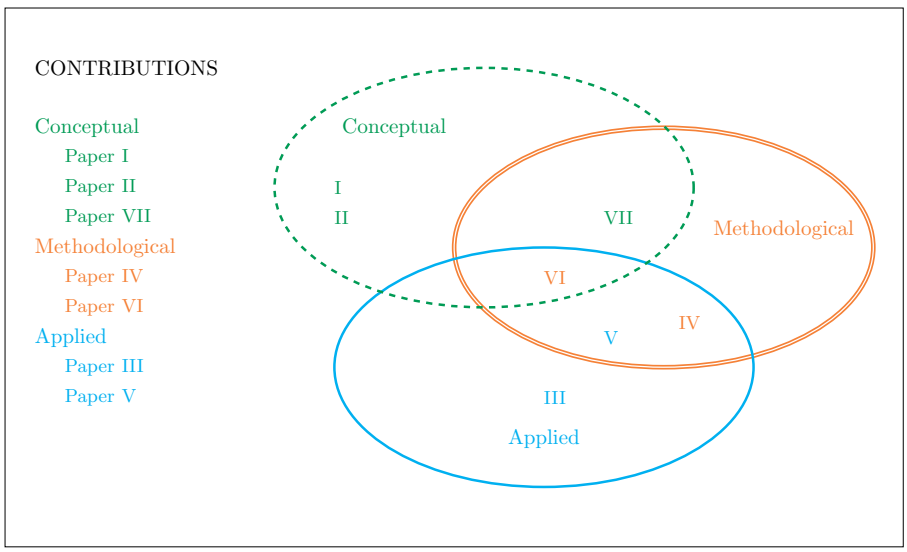


Figure 1.1: Visual map of contributions as a Venn diagram. The color coding indicates whether the contributions are *conceptual*, *methodological* or *applied*. The visual map indicates the affinity of each article for each other and relation to the three clusters. For instance, Paper IV is a methodological contribution but overlaps with the applied cluster, and is far away from the conceptual cluster. Several of the papers blend between different categories.

1.5.1 Conceptual

- Paper I, Paper II, Paper VII

This thesis has strongly focused on conceptual and theoretical contributions for musical live coding [35]–[37]. I aim to address the various qualities of musical gestures in live coding by observing performance systems that are notably different from one another. I introduce several terms to facilitate my discussion on theories of embodied music cognition and live coding, which is a powerful approach to communicating the novel properties of interactions that arise in musical live coding.

1.5.2 Methodological

- Paper IV, Paper VI

The methodological contributions rely upon the live coding ethos, which adheres to a free/open-source ethos. I present two studies, one exploring multimodal percepts when practicing live coding [38], and one conceptual framework for designing agent-based systems [39]. A central aim in both contributions is to help practitioners unclear on how to practice live coding and help them find their path by implementing advanced technologies in their systems.

1.5.3 Applied

- Paper III, Paper V

The applied contributions of this study reflect on practice and deliver open-source software to be used for creative and analysis purposes. In Paper III [40] we presented a redesign of our previous work on low-level programming languages for live coding [41], [42], and explored how gestural interactions can be used in a practical context. Novel technologies relatively unexplored in live coding practices, like the technical dimension of liveness [43], are discussed and connected to the creative potential of such technological advancements by means of delivering a software prototype. Also, a reproducible framework was developed for exploring musical form in live coding performances [44]. Practical implications for visualization technologies are presented, discussing their potential to be used for interactive applications.

1.6 Challenges in musical live coding

Live coding is becoming a 20-year-old practice this year, in 2023, as the first published article introducing the term appeared in 2003 [22]. There are numerous advancements and challenges, ranging from significant efforts to develop specialized programming languages for live coding, to novel systems extending interactivity, and collaborative musical ensembles that perform telematically. However, I mostly examine in this thesis several challenges which

are present today, addressing embodiment, creativity, risk as well as interactive AI issues.

Embodiment of musical interfaces has been an issue in open discussion since the early days of live coding [45]. This thesis strongly focuses on embodiment and contributes a theoretical understanding based on sensorimotor control and studies on embodied music cognition [30], [31]. Creativity also has been given strong attention within the community of live coders, and numerous interactions with other communities, such as CHI, Creativity and Cognition and more, extend beyond the limits of musical live coding in particular.

Of course, many more challenges in live coding are not directly addressed in this thesis. For example, collaborative live coding presents one of the hard problems in the field, although there have been numerous efforts and contributions since the early days of live coding, it is a field with technical, musical and philosophical challenges. Furthermore, issues on gender and accessibility in live coding are more topical than ever, as computing is largely still known as a western male-dominated research field. This thesis lacks any contributions to the topics of collaborative live coding and gender studies, and only touches upon issues of accessibility.

1.7 Research questions

Some research questions correspond to a specific study or a specific article, whereas others span across multiple studies and articles. In this thesis, a central research question was crystallized out of several years of practice and research on live coding. This is the first research question (RQ1) from the list below and relates to how we can live code when at the limits of our perception. This came out of experimentation while building musical interfaces for performance, where it became obvious that there is a need in music interaction to overcome received best practices used in human-computer interaction. The following research questions were formulated during the process of doing this doctoral research.

- RQ1: Are pre-reflective processes evident during a musical live coding performance?
  - What are the necessary conditions?
  - In which cases do these conditions occur?
- RQ2: How can generative algorithms influence musical gestures used during live coding performance?
  - How is this evident in live coding, and how is its appearance similar to or different from its appearance in work with digital musical instruments and other interactive music systems?
- RQ3: What creativity support technologies are used in musical live coding?

  - What is the role of liveness, and how we can use the technological advancements of liveness?

- RQ4: How do we practice musical live coding with assistive technologies?
  - What is the role of multimodal experience in live coding practice?
- RQ5: How can and do we design agent-based systems for musical live coding?
- RQ6: How is risk expressed in musical live coding?

The research questions are answered in conclusion, in the subsection "Research outcomes" (section 8.2).

1.8 Structure of the thesis

The thesis is structured in four studies (Study A, Study B, Study C and Study D). Each study is a synthesis of more than one of the appended papers, and each paper contributes to one or more studies. Table 1.1 presents the contribution of each paper on each study. A color-coding scheme based on the above mentioned categorization of **conceptual**, **methodological** and **applied** contributions is shown that aims to communicate the content of each study visually. The table below is inspired by Alan Blackwell's book Moral Codes[4], and aims to complement Figure 1.1.

Table 1.1: The table shows the contribution of the papers to each of the four studies. The relevance of a paper to each of the studies is indicated as low-relevance (`x`), mid-relevance (`xx`), or high-relevance (`xxx`), when relevant. A color grading is applied based on the relevance of each paper to each study.

Articles/Studies	**Study A**	**Study B**	**Study C**	**Study D**
Paper I	x	xx		
Paper II	x	xx		
Paper III	xxx		x	x
Paper IV			xxx	xx
Paper V			xx	xxx
Paper VI			xxx	
Paper VII	xx	xxx		x

- **Study A**: On-the-fly algorithms for machine musicianship
- **Study B**: Embodiment and musical gestures in machine musicianship
- **Study C**: Creativity support technologies for live coding
- **Study D**: Reproducibility, transparency and risk

[4]Alan's Blackwell book *Moral Codes* on PubPub: `https://moralcodes.pubpub.org/`, see Chapter 1: "Are You Playing Attention?"

Chapter 2

Theoretical background

The theoretical background is based on a phenomenological approach to live coding or unconventional programming practices. Phenomenology exploits lived experience [46], and here the work I present is influenced by notable philosophers such as Edmund Husserl, Maurice Merleau-Ponty, Jean-Paul Sartre, Humberto Maturana and Francisco Varela. It represents a transcendental phenomenological approach that focuses on understanding the practice of live coding as the play of sensorimotor processes unfolding during a performance.

2.1 Live coding 101

Live coding is an unconventional programming practice, sometimes also called *on-the-fly*, *just-in-time*, *conversational programming* or *live programming*. Unconventional programming practices range from amorphous and chemical computing to bio-inspired, autonomic and generative programming [47, p. V]. Essentially, live coding is "on-the-fly decision-making" [48, p. 14], and it may be necessary to make human-machine communication more fruitful and resulting in fewer unintentional outcomes. For instance, there has been more than one incident wherein AI systems have exhibited unwelcomed behaviours, and this could have been avoided if the systems were less rigid in their operation (e.g., Microsoft's Twitter bot[1] in 2016).

This thesis examines musical live coding systems and practices, mostly expressed as generative and autonomous processes. Generative processes may be rule-based, random, or algorithms that involve learning, such as machine learning and deep learning. The matter of autonomy in computer programs is explored in research on agent-based systems. The idea of "agency" is related to the role of the machine in live coding, or a sense of the machine's "otherness" [49]. The result may not differ greatly from other forms of electronic music, such as improvising with audio synthesizers, but live coding is also an improvisational practice. The sense of "otherness" can be seen as equivalent to the existence of co-creative processes, as a live coder co-creates the musical outcome along

[1] `https://en.wikipedia.org/wiki/Tay_(chatbot)` (accessed 29-08-2023)

with the contribution of autonomous processes run by the written program. In this thesis, co-creativity is explored from a phenomenological approach, which implies a first-person perspective of the human agent and a second-person perspective as an outcome of the relational role of the artificial agents that can induce this sense of "otherness" in the human performers, or of a *shared agency*.

It is widely acknowledged in the community of live coders that there is no school for learning how to live code. Paradoxically, the impact of live coding in education is already significant [50], [51] and more and more instructors employ live coding practices in programming education [52]. Live coding transcends the nature of conventional programming practices, expanding these practice to include a non-utilitarian programming perspective. The uncreative approach to organisational programming has been augmented to include a "recreational" approach of coding for fun [27]. Live coders do not adhere to the traditional practices of writing computer programs (e.g., pen and paper), and in so doing stretch what is known as agile programming to its extremes. Contrary to agile programming, there are no code requirements and no testing in place, but rather an *on-the-fly decision-making* process that incorporates various perceptual modalities. Thus, learning processes are become heavily focused on *trial-and-error* techniques, and learning paradigms like *learning-by-example* are at the core of live coding practices.

From a practice perspective, live coding has been addressed as a music-making activity that ranges from live instrument making [53] to an activity similar to conducting an orchestra [54]. While practices on live coding can widely differ, it is still the case that typing is the most widely used approach to live code. Thus, most performances are based on typing on a keyboard and re-writing programming statements. The written code is a form of prescriptive notation [55], or metaphorically speaking, can be seen as the genotype of the musical outcome [56], whereas the heard sounds are the phenotype of the musical outcome, that depend on uncontrolled variables such as the acoustic surrounding. In essence, the live coder, during a performance, is evaluating on-the-fly the written code by listening to the musical outcome, thereby assessing on-the-fly the heard musical outcome to adjust or modify the written code.

Typing is at the center of attention when it comes to the most widespread form of musical live coding: typing on a keyboard by sharing our screens in front of the audience. The code is the means for music-making, but the materiality of the code transcends into the quasi-material nature of algorithms [57]. "Algorithms are thoughts" as it is noted in the TOPLAP Draft Manifesto [2]. Thus, code is how we transform our thoughts into algorithms. The importance of written code itself is not paramount: in a recent talk Alex Maclean demonstrated a live coding session without any form of written representation[3].

This discussion relates to the ephemeral nature of code and the long-standing discussion of whether live coding is creating an oral tradition; we do not maintain every single line of code or every single code variation.

The main practice of musical live coding involves three main components:

[2]ManifestoDraft, `https://toplap.org/wiki/ManifestoDraft` (accessed: 2023-09-06)

[3]Alex McLean presents "live coding without anything" – `https://tinyurl.com/2s3r6s8w`

writing the code, listening to the musical outcome, and re-writing of the code. This is a feedback loop between writing and listening, expressing formal rules and experiencing what they sound like. Thus, the actual practice of expressing logical statements is transcended by the metaphysical nature of what they sound or look like, as in the case of visual live coding. Radically we may say that live coding is a transcendental bridge between what is known and what is experienced.

2.2 Historical accounts of programming during a performance

In this section, I present how technological advancements in the computing industry gave rise to a new form of electronic music, namely performing music using microprocessors, and how the deconstruction of the same infrastructure gave birth to live coding.

Programming during a performance is a natural evolution of live electronic music performance, which predates live coding practices by a few decades. The technological advancements in Silicon Valley during the 70s significantly influenced this era. At that time, composers were experimenting with affordable microprocessors in the San Francisco Bay Area[4]. Eventually, a KIM-1 microprocessor (Figure 2.1) became a *must have* which eventually led to the early experiments on collaborative computer music, most notably with bands such as *The League of Automatic Music Composers* and later *The Hub* [58].

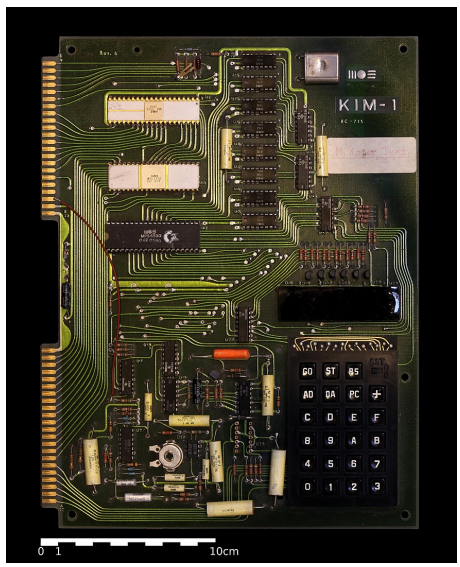


Figure 2.1: The KIM-1 microprocessor. On display at the Musée Bolo, EPFL, Lausanne. CC BY-SA 2.0.

[4] http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html

The prehistory of live coding was compiled as an online release named *TOPLAP001 – A prehistory of live coding* and released by wertlos.org (`http://wertlos.org/various/`). As live coding began with computer musicians, the immediate precursor of live coding were the pioneers of the field, like Ron Kuivila known to have performed live using a computer at STEIM in 1985, and The Hub and the League of Automatic Composers who were also performing in the Bay Area the same period. The League started in the late 70s and later some of its members formed The Hub. As early as 1979, they had begun regular biweekly informal public presentations at the East Bay Center for the Performing Arts, organising concerts with several associated members.

Live coding is tightly connected to the flourishing of generative arts that started in the 80s, and the community documents itself with a sense of irony and humour [57]. This is reflected in the title of the organisation for the promotion of live coding, called TOPLAP[5]. In the TOPLAP mailing list archives this sense of humour, or maybe the necessity for openness and self-criticism, is addressed in the very first posts[6, 7, 8]. Thus, a sense of extra-live-coding meaning is ascribed to self-reflections within the community. Alan Blackwell noted in his talk in PPIG 2022 that Alex McLean and Nick Collins introduced a methodology of "ironic deconstruction of the infrastructure". One of the main motivations for live coding centered around the criticism of DJ-ing using conventional media players to reproduce ready-made compositions, and live coding concerts had already been present in nightclubs in the UK [59].

2.3 From mnemonic devices, to interactive intefaces and on-the-fly algorithms

In this section, I continue with a more detailed history of algorithmic practices for music performance. I start by presenting algorithmic thinking and how the advent of computers shifted our understanding of musical processes from static to dynamic processes. My main concern in this section is to present how performers shift their understanding in correspondence with contemporary technological developments and how that has resulted in creative modifications of their algorithmic practices.

Algorithmic approaches to music composition can be traced back to the work of Guido d'Arezzo (circa 1026) during the medieval period in Europe [60], [61]. D'Arezzo's algorithmic rules primarily functioned as mnemonic devices, with the *vowel-to-pitch* lookup table of "AEIOU" mapped to pitch intervals serving as a notable example [62, p. 15]. The utilization of algorithms in musical composition has persisted for centuries, with the advent of information theory and digital computers in the 20th century providing new avenues for experimentation in the field.

[5] TOPLAP wiki Main page, `https://toplap.org/wiki/Main_Page` (accessed: 2023-09-06)

[6] Plain text file of the mailing's list archive from 2004-2008 `https://raw.githubusercontent.com/yaxu/unravelling/master/livecodemlarchive.txt`

[7] Complete mailing list archive: `https://toplap.org/livecode-archive/`

[8] `https://github.com/yaxu/unravelling`

One of the earliest known computer programs developed for music composition was the Illiac Suite [63], an advanced program that employed Monte Carlo algorithms and Markov chains. These probabilistic algorithms illustrated the potential for computer-generated music, despite not being the first instance of generative processes in the history of music-making (e.g., gamelan music [64]). Concurrently, stochastic processes for music composition were being explored using computers [65]. Famous composers like Xenakis were already experimenting with stochastic music [66]. During this period, the interaction between composers and computers remained largely static. The *composer-programmer* was limited to initiating the execution of a program and waiting for the output, which at the time was typically a list of integers mapped to musical notes.

As a result of the swift progress in computational capabilities and the miniaturization of integrated circuits, epitomized by Moore's Law, the age of personal computing emerged, making computer programming increasingly accessible to non-expert programmers. Scripting languages such as Perl and BASH facilitated rapid and interactive communication with computers by employing interpreters and runtime command prompts. This development eventually allowed for real-time modifications of computer programs, particularly in languages like LISP and later Smalltalk.

2.3.1 The first era of interactive music systems and on-the-fly algorithms

The advent of a new era in computer-assisted music-making facilitated the emergence of groundbreaking ensembles such as the *League of Automatic Composers* (League) and *The Hub*, during the nascent stages of networking and remote computer communication (the late 1970s - early 1980s). Initially, the League was experimenting with real-time data transmission between performers using solderless breadboard [67], [68]. Eventually this evolved to sending and receiving data over a micro-computer network system, called the "Hub", during musical improvisation [69], [70]. The first hub, called Hub 0.0, was transmitting data at a rate of 300 bits per second. The next version, Hub 1.0, reached a communication speed of 2400 bits per second. In 2004, Hub 3.0 was using an Ethernet router based on Ross Bencina's OSCgroups[9], a system for routing OSC[10] messages. As data and algorithms are often considered interchangeable concepts by some computer scientists, the Hub's achievement lies in allowing on-the-fly modifications of their programs for ensemble music performance, thereby illustrating the feasibility of computer-generated music within a social context.

Typically, a music psychology perspective holds an adaptationist account of the origins of music, which posits that music has survival value and is inherently social [71]. So, the accomplishment of making ensemble music with computers resonates with the essence of music as a collective activity. While solo performances present their challenges, the complexity of ensemble

[9] `http://www.rossbencina.com/code/oscgroups`

[10] Open Sound Control protocol, `https://en.wikipedia.org/wiki/Open_Sound_Control` (accessed: 2023-09-06)

performance is considerably heightened. As such, the psychology of music sees ensemble performance as an exceptionally demanding task and one of the most intricate human activities.

The transition from static algorithmic representations to dynamic interactions facilitated novel modes of musical expression. This development is at the core of interactive music systems, a wide-ranging research domain and the primary focus of the NIME conference. Rowe [2] characterizes interactive computer music systems as "those whose behavior changes in response to musical input," Zicarelli [72] portrays these environments as possessing "controls that offer immediate feedback in modifying an ongoing process." Such innovations have enabled the integration of computers into musical performances and facilitated direct interaction with the algorithms.

Rowe [2, chap. 1] raises the critical question of establishing meaningful communication between machine language and musical significance. How can low-level signal processes be translated into high-level, comprehensible musical representations for human understanding? This concept forms the underlying premise of machine musicianship. Rowe addresses this challenge by incorporating insights from music cognition studies, developing a sophisticated cognitive architecture that spans various levels of cognitive organization, such as hearing, memory, and meaning, to create tools for musical performance.

2.3.2 Towards creative modifications of algorithms

The initial phase of interactive music systems marked a significant milestone in the history of music-making, as these systems could produce unforeseen musical results, potentially fostering creativity during performances [73]. While such outcomes could be attained with mechanical systems, the programming capabilities in the age of personal computing facilitated swifter integration. Instead of reconstructing a mechanical system entirely, one could rewrite and recompile a software program — a process that, in many cases, would have been unfeasible with mechanical systems due to the temporal nature of music. Nevertheless, during the early days of computing, these processes were comparatively slow and could impede musical creativity.

Despite these limitations, musicians could interact with the systems in real-time. However, the systems could only respond within the constraints of their pre-programmed generative spaces. Here, an analogy may be drawn to contemporary AI algorithms trained on large datasets, where the training of a model may take anywhere from several hours to months. When interacting with such large models, adjustments are typically confined to the network's weights, without the ability to modify the training process on-the-fly, other than by applying reinforcement learning on top of the pre-trained large model to fine-tune its weights. This scenario bears similarities to the early days of interactive music systems and machine musicianship, which persisted until the mid to late 1990s.

As personal computers became increasingly powerful, real-time sound synthesis engines emerged, pioneered by the MUSIC-N languages [74] and the groundbreaking implementation of the *Unit Generator* (UGen). To this day,

UGens remain the primary tool for generating real-time sound synthesis engines. Music programming languages such as CSound, MAX, and OpenMusic provided new possibilities for performing musicians. Subsequently, PureData, an open-source version of MAX/MSP, and SuperCollider [75], transformed how musicians interact with musical programs in real-time. By the late 1990s, computers had attained sufficient computational resources to accommodate the requirements of real-time sound synthesis for music-making. At this point, it was not far away that real-time sensing technologies, such as online machine listening to the audio [76] became available.

2.3.3 The early days of live coding

In the early 2000s, personal computers and laptops gained popularity in on-stage musical events, from DJ sets to improvised performances. This trend often led to the non-creative utilization of computers, relegating them to a function similar to other music reproduction devices, such as digital compact discs (CD-ROM). In response to this development, live coding emerged as an "ironic deconstruction of the infrastructure" as Alan Blackwell noted in his PPIG presentation [77], gaining traction on stage and in mailing lists.

Motivated by a blend of humor and technological curiosity, the live coding pioneers questioned the nature of computer interaction during musical improvisation. They criticized mainstream commercial software interfaces like Ableton and Reason as rigid and restrictive interfaces [22]. Instead, live coding practices emphasized malleable, non-rigid interfaces and democratic approaches [78] to music-making. The Powerbooks ensemble, for instance, comprised musicians performing with laptops on their laps, scattered among the audience, and utilizing their laptops' built-in speakers.

During the first decade of 2000s, the popularization of the *maker culture* emerged in parallel, characterized by advancements in open hardware such as the Arduino microcontroller. Precursors in music include Nicholas Collins who was developing DIY electronic musical instruments [79] since the early 1980s. While the two communities overlapped to some extent, the maker culture garnered broader attention and societal impact, possibly due to the tangible outcomes of open hardware, vast interdisciplinarity and capitalization opportunities of the innovation. In contrast, the early days of live coding were primarily confined to musicians' circles. Presently, live coders constitute an international community with active local hubs worldwide[11]. The art form has expanded beyond music to encompass dance[12], visuals[13], mechatronics[14], weaving[15], and more.

Live coding serves as a prime example of on-the-fly interaction with algorithmic processes. Although the practice does not significantly deviate from a blank-slate patching session using modular synthesizers, live coding goes a step

[11] Currently 38 nodes worldwide `https://toplap.org/nodes/` (accessed: 2023-09-05)
[12] HTB2.0 - Kate Sicchio Concert D, Tuesday `https://youtu.be/iOAffWTBVE0`
[13] Olivia Jack - Hydra, Live Coding Visuals in the Browser `https://youtu.be/cw7tPDrFIQg`
[14] Live coding techno with a turntable `https://youtu.be/IatfZAIMU_M`
[15] CODE • WEAVE • MUSIC `https://youtu.be/rR1o_IytPl8`

further by redefining and questioning its own definition (if any)[16]. Analogous to a modular synthesizer, live coding posits that any module is subject to redefinition, on-the-fly.

This shift in understanding can be likened to the static mathematical representations we impose on ourselves when doing (or undoing) mathematics. The inadequacy of static mathematical formulations was a topic of debate in the early days of phenomenology, notably in the discourse between Husserl and Brouwer [80], [81]. Viewing mathematics as a static entity effectively denies the experience of understanding mathematics and any potential re-understanding during the process. While some may perceive this as time in mathematics, others might consider it the journey of mathematics. Live coding essentially incorporates temporality into the process of programming [25].

2.4 Live coding as a musical activity

Here, I present live coding as a musical activity and see three main components: musical practice, musical agents, and musical systems. The musical practice has been discussed in live coding and is seen as the main vehicle for learning how to live code. Musical agents, here, refer to both human and machine agents, and I discuss aspects of agency from the live coding literature. Finally, I discuss live coding systems, which seem to be a necessary component, comprised of either mechanical, software, or notational constructs.

2.4.1 Live coding practice

Traditional musicianship requires much practice, and the same applies to live coding [38], [54]. In music technology, sometimes we forget that practice can improve skills, and music performance is a skillful human activity. To a certain extent, this may be linked to lower accessibility requirements when starting to use a digital musical instrument (DMI), a digital audio workstation (DAW), or a program for editing sounds like Audacity – one of the most popular and open-source sound editors. The software music industry has been focused on making their products more and more accessible, and maybe the epitome of this is realized with the prompt engineering AI applications which simply spit out generative compositions based on style imitation algorithms. As of early 2023, there are numerous online text-to-song generators, but as an example I will point out Jukebox[17] developed by OpenAI, the company that made a difference with their chatbot, called *ChatGPT*, in 2022. Although I will not analyze this situation, I will make an analogy to the era in which sound recordings first

[16]There is an endless discussion about whether live coding requires a definition. In this thesis, I deliberately avoid entering into this discussion, and I adhere to the good faith of live coders, acknowledging as live coding any systems and practices that use code for live performances. Several attempts have been made through the years, most notably in the first ICLC in 2015 where the participants were asked to define what live coding is `https://docs.google.com/spreadsheets/d/1ogux4mAIUbXOjiFqw7fhh6WL0FKHdVvvuijEeZl5jCM/edit?usp=sharing` (accessed: 2023-09-05).

[17]`https://openai.com/blog/jukebox/`

appeared, which have been criticized substantially for transforming the social nature of music. On the other hand, recording technologies enabled archiving and personalized listening activities. How the transition from personalized music listening to a *personalized music corpus*[18] will impact our musical activities and the social nature of music is yet to be seen.

Regardless of the above advancements, musical practice is an activity that typically involves both our mind and body in the situation. Musical activities can be conceptualised as consisting of *music-making*, *music listening* and *musical imagery*. Music-making involves both music composition and music performance. Music listening is the most widespread musical activity. We are exposed to music listening many hours per day, even involuntarily. Musical imagery is related to our ability to imagine musical melodies and can be both voluntary and involuntary [82]. When practising music we engage in all three above activities, which indicates that these activities are *not* mutually exclusive.

On the contrary, the three above-mentioned musical activities can be seen as progressive levels of engagement with music. It's simply self-evident that we are more engaged when we play music than when we listen to music. For example, we can listen to music and engage in a conversation, whereas the same cannot hold when we perform music, as we are bodily engaged in the music production. Similarly, when listening to music, we are more engaged with the music than when we merely imagine a melody of a song. It can be hard to start dancing to the beat while only imagining the music, whereas when we listen to dance music, we are likely to get bodily entrained to the beat. Overt bodily movement indicates increased engagement with music, as our body is linked to the environment, regulates our organism, and so on. Essentially, the more embodied our interactions with music, the more engaged we are with the musical activities.

All three musical activities are involved in live coding, similar to traditional music performances. We typically make some effort when writing the next code snippet, which involves processes related to musical imagery. Indicatively, notational audiation [83], [84] may be activated during live coding [37], as code is the notation in live coding [55]. Musical imagery also contributes to planning sequential actions [85], and typing on a keyboard is an activity based on *serial skilled actions* [86]. Of course, there is a difference between serial skilled actions in traditional music performance and serial skilled actions in live coding, as "the code writing of deferred time computer programming may be assembled out of time order" [87, p. 115]. Whereas traditional musicianship is based on temporally precise musical gestures, and musicians are considered the "champions" of sensorimotor synchronization.

Nick Collins and Fredrik Olofsson conducted a month-long daily practice using SuperCollider [54]. I also conducted a month-long daily practice study examining how listening to the generated sound can influence the coding session [38]. To the best of my knowledge, there are no other studies that follow the daily practice paradigm. What is obvious from both my study and Collins-Olofsson's study is that daily practice improves our live coding

[18]With *personalised music corpus*, I refer to the possibility that each music-lover can potentially generate his or her own "favourite" song list that meets their musical preferences.

skills. Knowledge and skill are seen as interchangeable notions for Satinder Gill and intertwined with skilled performance [88]. In musical live coding, besides any musical knowledge, there is a requirement for some basic programming skills. Collins also suggests several exercises, some demonstrating a hilarious attitude for radical experimentations, but certainly some very good points. The exercises are separated into three sections *isolation exercises*, *connectivity exercises* and *repertoire implications*. Isolation exercises may not be related to exercising musical or programming skills, as typing on a keyboard does not fit in either of the two categories. Connectivity exercises address practices that can be used to build a battery of live coding skills that can be ready at hand during a performance to make transitions between musical structures and programming abstractions. Repertoire exercises involve practising repertoire, which in live coding, at this point there is likely no such thing yet.

2.4.2 Agency in live coding

Agency in live coding has been an issue of discourse since the early days [23]. The agency is seen as distributed between the human and the machine. Andrew Brown discusses the potential controversies in ascribing agency to non-living things and suggests the term "relational agency" be used to describe artefacts and machines [49]. Thus, a "black box" machine may not have agency, but as part of human-machine interactions, the machine exhibits relational agency. Brown argues that machine musicianship is not fundamentally different from traditional musicianship, as some of the basic mechanisms responsible for social bonding and music engagement, such as rhythmic entrainment, are present. What is different in machine musicianship is an "enhanced sense of agency or otherness" [49, p. 184].

Dahlstedt [89] introduces the term *influential agency* to describe how the designer of a tool has influence over the artwork. He sketches a *spectrum of agency* (Figure 2.2), which shows how agency progresses from the designer of a tool to all involved parties. In live coding, we can see that in programming languages, such as TidalCycles, where the designer of the tool, in this case Alex McLean, has a certain degree of influence over the final artwork, as the artist has to rely on pattern structures embedded in the language for the final composition/performance. A similar notion is that of the *idiomaticity* of a programming language [90], a term to denote how different programming languages can influence the musical outcome.

2.4.3 Live coding systems

The music tradition that enabled practices like live coding to appear builds upon automated musical instruments like the barrel organ and music boxes [49]. The diversity of live coding systems is paramount, as this has already been addressed in the diverse practices that resist any easy forms of classification [25, p. 231]. Whereas systems like TidalCycles lowered the entry level for live coding, and made sense for making algorithmic dance music, it can still be difficult to say that two live coders are using the same system, as the system is

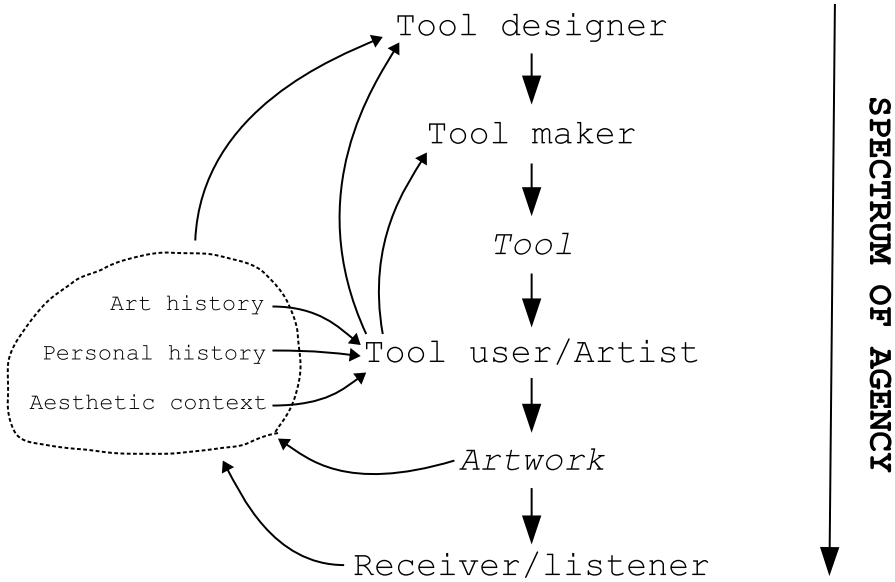


Figure 2.2: A skectch of the *spectrum of agency* demostrating visually how the designer has a degree of *influential agency* over the artwork. (Illustration from Dahlstedt [89]).

being extensively modified in use, on the fly.

On the other hand, some sort of live coding system seems necessary, although it may be something as simple as notation or as complex as a computer architecture or a distributed infrastructure. The code itself is seen as a notation in live coding practice [91]. There is a long tradition of relations between notations and systems, ranging back to Guido d'Arezzo and mnemonic rules [61]. As such, notation in live coding is seen as an epistemic tool and, thus, an extension of our minds [92]. I see no difference between mind and body, and regardless of whether code vibrates or not[19], as was discussed by Nicholas Collins [93] and Thor Magnusson [92], I see the notation as an extension of our embodied understanding [94].

In recent years, the newly established forum *Hybrid Live Coding Interfaces*[20] is gaining attention. In this annual event, the discussions focus on hybrid systems that extend beyond the keyboard and, usually, the screens. The systems presented may range from piano and other physical instruments coupled to a live coding interface, to virtual reality and speculative visions about the field.

[19]We have demonstrated in previous work, how quasi-palindromic structures emerge from generative algorithms [42], and use them in practice. I assert from musical practice that a certain level of anticipation is to be met when the generative rules are met with the compositional rules. This simply means that the performer can anticipate the quasi-palindromic structures by steering the system, and it is unclear to me whether familiarity is important here. I, here, refer to the spectrum of generativity by Dahlstedt [89], where the *generative algorithm* converses with the *rule system* of the compositional rules.

[20]`https://hybrid-livecode.pubpub.org/`

Chapter 3

Study A: On-the-fly algorithms for machine musicianship

> "The code is like clay and I am sculpting it."
>
> Fredrik Olofsson [95, p. 37]

In Study A, I present how live coding contributes to a shift from *cognition through perception* to *cognition through interaction*. The theory behind this will be further elaborated in the next chapter (Study B) and can be seen as an extended discussion on this topic commonly referred to as action-perception theory [96]. In the present chapter (Study A), I focus on the technical developments in the field that enabled the flourishing of such an understanding. Then, I continue on the system side and discuss the technical dimension of liveness in programming environments, as introduced by Tanimoto [43]. I discuss how liveness is also used to describe percepts and experiences. I continue with two main strategies of live coding based on a cognitive information processing paradigm. The subsection *Bottom-up systems in live coding* (section 3.3) presents a survey of all known systems that follow this approach to live code. The last two subsections of this chapter focus on my contributions and their implications.

3.1 Live coding in machine musicianship and software engineering

3.1.1 In machine musicianship

Live coding can be characterized as altering a program while running [18], [22], [23], [59], [97]. Within the realm of music, coding translates into sound, allowing the composer-programmer to experience the written program audibly while formulating the governing rules. The running programs are perceived as "live" primarily because the resulting output is experienced as audible

sounds. Our cognitive capacities enable us to form this perceptual continuity by grouping musical elements using cognitive processes [98], such as long-term and short-term memory. Long-term memory (LTM) is responsible for structural elements, such as musical patterns, themes, and motifs, whereas short-term memory (STM) is responsible for melodic and rhythmical groupings [99].

Gestalt principles, such as proximity, similarity and good continuation are some of the important mechanisms that contribute to grouping and the emergence of music perception. How we progressed from an auditory perceptual understanding to making semantic relationships between written code and heard sounds is not fully understood, but the essence lies in the interaction between code and sound as formulated by our lived experiences. In this thesis, I later discuss[1] the importance of musical imagery and music listening as coupled to our sensorimotor network and the possible influences of notation and sound with bodily movement [37].

3.1.2 In software engineering

Tanimoto [100] introduced a hierarchy of liveness for programming environments. The hierarchy had four levels: informative (L1), informative and significant (L2), informative, significant and reactive (L3), and informative, significant, reactive and live (L4). This will be referred to as the technical notion of liveness, as it pertains specifically to programming environments and does not encompass the social context of musical performances. Two decades later, Tanimoto [43] revisited his hierarchy, incorporating two additional levels. Algorithms could now anticipate user behaviors, also known as *tactically predictive* (L5), and make informed decisions about the programmer's intentions, also known as *strategically predictive* (L6). L5 and L6 will be referred to as *advanced levels of liveness*.

The concept of *liveness* in musical live coding is an intrinsic attribute of systems during live performances [101]. This characteristic is not contingent on edit-triggered updates, as outlined in Church et al. [102], as in a performance context, live coding systems make use of stream-based code evaluations[2] [101]. This extends beyond the technical definition of liveness introduced by Tanimoto in the context of programming environments because the multimodality of musical live coding allows different sensory channels to influence our experience. So, auditory and music perception have an active role in musical live coding that blends in its *many liveness-es* [25].

The evolution of the concept of liveness has led to algorithms being more than just dynamic entities that can be updated on-the-fly. The advanced levels of liveness realize the notion of the potential for multiple future outcomes within programming environments. This is because the user can have a preview of possible future outcomes and select one or ignore all of them. Generative

[1] See Study B: Embodiment and musical gestures in machine musicianship

[2] There is a difference when systems use clock-based updates, where the code evaluations are imposed forcefully by the positive edge clock. This attribute is typically related to a "safety" attribute, that systems cannot run into error-prone evaluations, something I will elaborate on more in the sections Study D and Discussion.

art has been moving in this direction since the 1960s. Still, it took more than 30 years for artists to realize concept of multiple future outcomes, with the performance by Oliveros over telephony lines [103], and almost half a century to experience this during the unfolding of their creative practices[3]. Occasionally, interfacing with multiple future outcomes in live coding – such as in code previews – is perceived as an unwelcome feature in live coding, as our cognitive resources may not accommodate such increased amount of information flow [104].

3.2 Two approaches to live code

Two primary approaches to live coding exist: employing high-level abstractions to generate music in a top-down manner or utilizing low-level programming constructs to generate music in a bottom-up manner [40]. An example of the high-level approach to live coding involves typing on a keyboard and evaluating high-level programming statements. For instance, during a musical live coding session, one might invoke a high-level abstraction to generate a musical pattern, such as a musical scale, or produce a sinusoidal oscillation with specific parameters. Conversely, adopting a bottom-up approach entails constructing the levels of abstraction progressively and on-the-fly. For example, using an instruction set to write the program entails accessing opcodes[4]. To determine whether data reading access is provided, an arithmetic operation is performed, or results are stored in memory. In this way, different levels of abstraction are created and validated on-the-fly, as the incoming data are assigned to different functions determined by their corresponding opcodes/operands. Ultimately, bottom-up systems depend on a minimal use of abstraction.

From a cognitive perspective, top-down approaches to live coding typically involve long-term memory processes, as the time required to type an executable command in most programming languages exceeds the upper limits of short-term memory (indicatively 0.5-8 seconds [99]). Although some languages like ixi-lang were specifically designed to account for such cognitive constraints, as the time required to type an executable command in ixi-lang is less than 5 seconds [105]. TidalCycles also has an elegant and compact syntax that enables activations within the STM window[5]. Contrary, bottom-up systems typically involve a shorter duration between code updates, which can often involve our STM during a performance, as I will discuss later in detail.

3.2.1 The case of bottom-up systems

George Dyson, in his book *Analogia* [106], employs an apt metaphor from canoe construction. A canoe can be built using a top-down approach, as in the case

[3]Indicatively, the performance *40 Days 40 Nights - True Rosaschi*, where Pauline Oliveros plays the accordion via the internet – `https://www.youtube.com/watch?v=2qBI9Pdk_D4`.

[4]An *opcode*, also known as computing syllable, is an abbreviation for *operation code*, and are elementary machine instructions and elementary notions in computer organisation and architecture.

[5]Alex McLean live coding demo `https://youtu.be/PeyE8ATMezs`

of a dugout canoe, or a bottom-up approach, exemplified by the construction of a birch bark canoe[6]. This bottom-up canoe construction highlights the advantages conferred by the modularity of the construction components. In contrast, some of the limitations imposed by top-down architectures may hinder the construction process. If an erroneous step is taken while creating a dugout canoe, the material might become unusable[7]. The same does not apply to bottom-up systems, which exhibit inherent generativity as this is engraved in the modular design of such systems. In musical instruments, the same analogy can be made with the construction of the sound box (body) of an oud[8], and the sound box of a Cretan lyra[9]. The sounding body of the oud is typically constructed using small thin pieces of wood, whereas the sounding body of a Cretan lyra is made out of a single piece of wood (i.e., not modular).

As early as 1993, Blackwell presented a *bottom-up manifesto* [107], drawing inspiration from object-oriented programming (OOP) practices. The concept he advocates is not novel for computer scientists; rather, he emphasizes the significance of low-level abstractions that can enhance programming routines in daily tasks. In essence, top-down paradigms assume the existence of some bottom-up constructs, which ultimately become "invisible" to the user (this thesis does not differentiate between users and programmers, viewing users as experts).

Zmöling [108] suggests another view of the top-down and bottom-up paradigms as related to code comprehension [109] and consequently to audience perception of the code. He sees that the top-down approach has a predetermined goal of what the program is supposed to be doing, whereas the bottom-up approach aims to relate the novel code chunks with previously running code chunks. How the audience perceives the written code is not straight-forward but depends on various factors that will be discussed further in Study D.

Live writing, the process of writing text or a computer program, is also a bottom-up approach [110]. This practice can be expanded to the live writing of the parsing rules during a performance, a feature of the Sema ecosystem [111]. This rewriting of the parsing rules on-the-fly is, in principle, also applicable to some low-level programming systems, such as Betablocker and Al-jazzari [112]. The parsing rules of a programming language can be dynamically modified on-the-fly, which would generate more new languages. Some of them may not be useful at all, but some may enhance the creative potential of a system during a performance. For instance, we can create numerous mini-languages throughout a performance and investigate which ones yield more pleasing or dissonant musical outcomes. Such practices reshape the generative space of potential outcomes. The question arises: can AI architectures benefit from such bottom-up structures that examine multiple formal languages on-the-fly?

[6]Wikipedia Canoe, `https://en.wikipedia.org/wiki/Canoe` (accessed: 2023-09-07)

[7]Interestingly the same applies to top-down live coding systems, whereas in constrast bottom-up systems do not exhibit such attributes of failure, as their design is implicitly "safeguarding" the continuation of the running processes. Something similar applies to live patching using sound synthesizers, as the sound is typically playing continuously.

[8]Wikipedia Oud, `https://en.wikipedia.org/wiki/Oud` (accessed 2023-09-03)

[9]Wikipedia Cretan lyra, `https://en.wikipedia.org/wiki/Cretan_lyra` (accessed 2023-09-03)

3.2.2 A bottom-up methodology

3.2.2.1 Bottom-up practices in software engineering

Bottom-up approaches to programming have been explored both in software engineering (SE) [107] and in live coding, most notably with the use of live writing practices [110]. Historical perspectives in SE have generally promoted programming practices with a hierarchical and top-down approach. Blackwell, Cox, and Lee [110] discuss the high cost of programming during the early days of computing in business environments and how that influenced programming development culture. This emphasis on rapid and efficient programming, or *extreme programming*, ultimately led to the emergence of agile software development.

> "The top-down approach to software development, characterized by iterative refinement, stands in stark contrast to the alternative, bottom-up construction method. In this latter model, individual components are created and verified before being assembled to form a functional whole. The mindset of the 'bottom-up' developer aligns more with a pragmatist or bricoleur, emphasizing craftsmanship over the strict proof often associated with the steps of iterative refinement. Live writing performances inherently possess a degree of bottom-up methodology." [110, p. 2]

The bottom-up approach to programming opposes agile programming practices, which do not set requirements during the software development cycle planning, but rather aim for *requirements discovery*. Agile programming is rooted in object-oriented programming (OOP) practices, where "iterative refinement" is the primary problem-solving method. In contrast, bottom-up programming practices involve the progressive development of various separate modules by building and verifying their functionality. This approach is often seen in digital circuit design, where computer engineers engage directly with the physical materials of a computer architecture. In live coding, a bottom-up approach tends to emphasize material practices, as has been literally showcased by Reus in the iMac Music [113] performance and the stateLogic machine [41]. While not all bottom-up systems have a direct connection to hardware, there is often a strong link to the craft of programming due to the algorithm's inherent quasi-materiality within the system. Though iMac Music and the stateLogic machine are hardware implementations, the stateLogic machine was later written as software for SuperCollider. This connection typically exists even in systems *not* built directly in hardware, but even when implemented on a virtual machine.

3.2.2.2 Misconceptions between bottom-up and low-level processes

Low-level computations are not inherent components of a formal system. They can be employed to filter a digital signal or write a computer program. While filters are engineering abstractions that can be applied on a sample-by-sample basis, they do not rely on a formal grammar. The term *bottom-up live coding*

is introduced to differentiate between low-level processes using engineering abstractions and those operating on a formal grammar. Digital filters can range from simple linear signal transformations to complex, non-linear processes that create chaotic oscillations. Contrary, certain formal languages can be used for universal computations.

An exception to the abovementioned distinction between low-level and bottom-up terminology, is the Betablocker version by Bovermann and Griffiths [114]. In this specific version, Betablocker is the sound engine and the coder is typing high-level abstractions in SuperCollider. Thus, the sound engine relies on a bottom-up system and does not function similarly to a digital filter. This system essentially traverses from a top-down approach to typing on a keyboard to a bottom-up approach for making sound synthesis. It demonstrates in this manner how hard it can be to make easy classifications in live coding [25].

3.2.2.3 Live writing as a bottom-up practice

Live writing presents another case of bottom-up systems [110]. Till Bovermann presented a duet live performance with Sara Hildebrand Marques Lopes at Karlsruhe's *live.code.fest 2013*[10], where they performed with the *Oulipop* system. The performance was live writing, where the performers shared one keyboard and took turns in a live writing exercise that looks like *code poetry* [115] although the Oulipop performance consisted of prose [48]. The performance system is based on Betablocker's sound engine, and each keystroke manipulates a fictional computer architecture[11]. In this manner, live writing was implemented as single-keystroke commands. Later on, Lee and colleagues introduced the term *live writing* as the "asynchronous playback of a live coding performance" [116]. In 2016, Lee presented live writing as a performance at NIME[12] [117], where the rules are not based on single-keystroke commands.

3.3 Bottom-up systems for live coding

Here, I will present various bottom-up systems and briefly explain how their algorithmic processes unfold over time. I present the systems in separate paragraphs, and their corresponding videos in Table 3.1.

Within the various practices of musical live coding, there exists a limited number of systems that adopt a bottom-up methodology. The earliest documented instance of such a system is Betablocker, by Dave Griffiths, which employs a video game interface for live coding musical compositions [112]. Subsequently, Al-jazzari (Figure 3.1) was developed as a successor to Betablocker,

[10] Oulipop performance at live.code.fest 2013, Karlsruhe, Germany. `https://podcasts.apple.com/us/podcast/till-bovermann-sara-hildebrand-marques-lopes/id821590153?i=1000411352850` (video not available on 2023-06-25).

[11] "Manipulating texts according to Oulipo rules. Each keystroke triggers that the text's current state is translated character by character into a set of commands for a fictional CPU. Treated as a sound signal, the ever-changing output of that CPU is played back us and to the audience, guiding our decisions on how to alter the text next." (`https://research.aalto.fi/en/publications/oulipop` – accessed 2023-06-25).

[12] Live writing `https://youtu.be/1WRn2LNV9yw`

presenting a three-dimensional video game environment wherein animated robot agents visually represent musical occurrences. Both systems share a similar design, interfacing directly with the instruction set of the computer architecture.

Figure 3.1: On the projection there are four window frames showing Al-Jazzari in action, by Alex McLean and Dave Griffiths. Image by Dan Stowell is licensed under CC BY-SA 2.0.

A somewhat similar approach to Al-jazzari and Betablocker is the *TOPLAP app*, created by Nick Collins [118], which employs a "timbral instruction set" for live coding musical outcomes. The TOPLAP app moves further and implements a machine listening algorithm on the musical outcome. This operation serves as the driving force of the timbral instruction set by combining the user's input and the result of the machine listening analysis. In this manner, Collins speculates how we can write computer programs using the musical outcome of the system in question.

We have introduced a collection of live coding interfaces [40]–[42] that progressively advance the levels of abstraction, from bits to symbols and subsequently from symbols (bytes) to tokens (words). Essentially, the user provides the input in machine language, which is a binary representation of instructions, and the output is provided in a symbolic representation. Initially, I implemented this as a hardware prototype, the stateLogic machine, and presented it in the SuperCollider Symposium 2012 Sonicarts exhibition in London, UK (Figure 3.2).

Figure 3.2: The figure shows one of the two stateLogic machine prototypes that were presented at the SuperCollider Symposium 2012's Sonicarts exhibition. Photo by Steve Welburn.

Later we presented this approach as *live hardware coding* with Ioannis Zannos [41]. In 2014, a lexical analyzer was implemented on top of the stateLogic machine, and the algorithm design was able to recognize regular expressions. In 2022, we redesigned the input interface by employing a feedback loop between the encoding of the symbols and the input interface (in bits), facilitating direct manipulation[13] further by continuous gestural interaction (Figure 3.3). The user interface is redesigned in such a manner that the representation is not exhaustive, and we exchange *experiences of meaning* with aspects that can facilitate *interactivity*, like the *fluidity of actions* [119].

CodeKlavier [120], is a project by Noriega and Veinberg where the main point is that the pianist is programming using the piano. They refer to this dual functionality of the pianist by calling them "the piano-coder." CodeKlavier has been presented as a series of submodules, each implementing a different style of performance, from assigning a pianist the action of typing on a keyboard, to specifying audience interactions with an AR application. Here, I will focus on the CKalcuλator module, which can be categorized as a bottom-up system. The

[13]Direct manipulation (DM) is used when designing computer interfaces to facilitate direct control of actions, typically through continuous control, like in the case of using the mouse cursor to move a file to another folder. Another important aspect of DM is that it facilitates cognitive processes related to recognition instead of retrieval. Some would argue that an important component of DM is the *undo* functionality, but in the case of music performance this is not applicable; we cannot unplay a note.

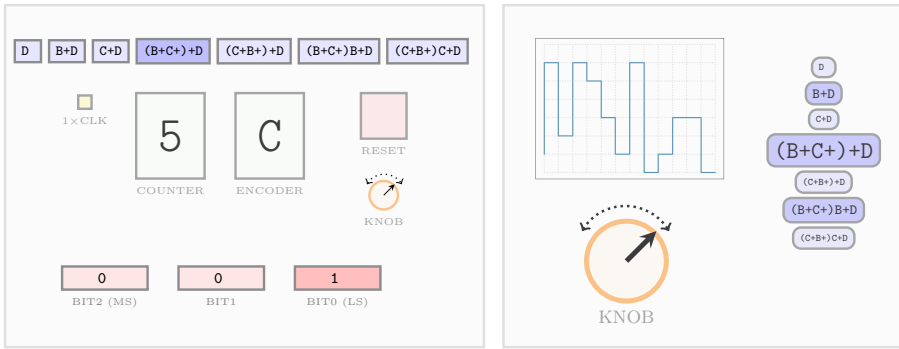


Figure 3.3: Left-hand side shows the GUI from the first revision of the prototype in 2014, that we introduced a lexical analysis module. Right-hand side show a preliminary sketch of the redesign that affords direct manipulation using a knob. Video demos: `https://youtu.be/AA78JewyU44` and `https://youtu.be/FZG_c1zKiVQ`

system recognizes melodic sequences by the piano-coder and constructs simple arithmetic operations on-the-fly by employing lambda functions, a common functional programming construct.

Jonathan Reus has notably adopted a more radical approach with iMac Music, in which he intervenes directly on the motherboard of a personal computer. This performance system is metaphorically referring to how we can gesturally intervene in the very workings of the computer. Jonathan Reus nicely demonstrates the importance of the craft and physical labor that the musician puts in during live coding. This approach contrasts with the typical approach to live coding, where the coder is just sitting at a computer. Reus's approach, where he is physically manipulating circuitry, is certainly a highly idiosyncratic system not unlike traditional modern music practices of *inside piano* playing [121].

Table 3.1: Table of bottom-up systems in chronological order.

Author	System	Year	Video URL
Griffiths	Betablocker	2006	`https://vimeo.com/24390484`
Griffiths	Al-jazzari	2007	`https://youtu.be/Uve4qStSJq4`
Diapoulis	stateLogic machine	2012	`https://vimeo.com/43121821`
Reus	iMac	2012	`https://vimeo.com/205714278`
Diapoulis	Low-level live coding	2014	`https://youtu.be/AA78JewyU44`
Collins	TOPLAPapp	2015	`https://youtu.be/hfJTF3KTnFM`
Noriega & Veinberg	CodeKlavier CKalcuλator	2019	`https://youtu.be/hD-PWNDebD4`
Diapoulis	Bottom-up live coding	2022	`https://youtu.be/FZG_c1zKiVQ`

A common attribute among the systems above is the capacity to construct abstraction levels dynamically and progressively. For example, the stateLogic

machine processes the 3-bit input in binary notation, progressively decoding and encoding it into symbols, providing the raw material for the lexical analyzer that recognizes regular expressions. Betablocker, Al-jazzari, and the TOPLAP app are stack machines, thus, are progressively implementing the computer program by assemblages of opcodes and data. CodeKlavier CKalcuλator implements lambda calculus on-the-fly by implementing simple arithmetic operations. For instance, a certain musical pattern corresponds to the addition operand, another to a specific number and so on. Finally, iMac Music presents a metaphorical design where the live coder implements abstractions by creating short circuits on the computer's motherboard. Thus, the performer is not modifying the software, but the hardware [122]. Reus's radical approach transcends the meaning of live coding in its every manifestation. The system is not "used" as intended to, but rather "misused" to bring forth aspects of craftmanship and labor during live coding.

3.4 Contributions

Here, I present the contributions of Study A in three categories: *user interaction*, *systems*, and *mapping*, and I end with a simple example to demonstrate the underlying algorithm of the stateLogic machine. My main contribution with this study centers around the statelogic machine and its follow-up versions. To explain how the system differs but also aligns with other live coding systems, I introduce the term *bottom-up live coding* [37]. The term is used to denote a live coding strategy that technically differs from the standard paradigm, sometimes also called *canonical live coding* [104], to denote the practice of typing on a keyboard using a programming language during a live performance, and sharing the screen with the audience. This bottom-up strategy was discussed previously (sections 3.2 and 3.3) with various examples and accompanied video demos. Essentially, the bottom-up system I present is a blend of data generation and algorithmic processes that can generate musical structures on different levels (micro-structure, meso-structure, macro-structure). I apply the algorithm to musical events, but in principle it can also be implemented on the signal level, for instance as wavetable synthesis (see the source code `https://gitlab.com/diapoulis/lhc-knob`). Also, the latest developments implement a gestural controller, which opens further possibilities as it presents a tactically predictive system in Tanimoto's hierarchy of liveness [43].

3.4.1 User Interaction

The primary objective of our approach to bottom-up live coding was to make live coding more "humane" by significantly reducing the temporal gap between registering input commands and experiencing the resultant musical output. To this end an event sonification is used to audify every step in the process (video demo `https://youtu.be/AA78JewyU44`), an attempt to solve the problem known in music psychology as *delayed auditory feedback*. Live coding, like most electronic music, already alters the meaning between produced gestural

actions and sound outcome, something we audibly experience as *altered auditory feedback* [123].

I have been previously experimenting with interactive interfaces by employing combinational digital circuits, but the stateLogic machine presented one of my first endeavors to devise a sequential digital circuit. Sequential circuits introduce a memory component in digital design by implementing finite-state machines and digital clocks. Using memory components for input control is common when interfacing with an electronic musical instrument. However, the approach of directly interfacing with a finite-state machine is unusual, particularly in industrial designs (i.e., commercial hardware and software interfaces). For instance, a typical sequencer[14] interface with 8 steps is shown in Figure 3.4. The input buttons can register values to the running sequence and correspond to the output using highlights. In this case, there is a 1-to-1 mapping between the input registered values and output sonic outcome.

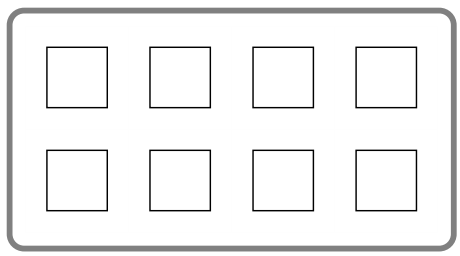


Figure 3.4: Simple sequencer with 8 input buttons.

A visual representation of the basic algorithm for the 2014 prototype [42] is shown in Figure 3.5. (The same algorithm was used in Paper III [40]). In this case, the user is interfacing with three input buttons[15] (Layer 0), and there are three layers corresponding to the output (Layer 1, Layer 2, Layer 3). Thus, the input and output do not have 1-to-1 relation. Instead, the levels of abstraction are progressively built up, which increases the complexity of the system dramatically. The main difference for the user compared to a simple sequencer is that here the user has to construct a more complex *mental model* [124] of the behavior of the system to predict how the system will behave. Thus the anticipation of the future outcome requires multiple levels of abstraction, as opposed to a simple 1-to-1 mapping interface, where each input command corresponds to a single resultant musical outcome.

In terms of musical practice, the main difference in how the user interacts between a bottom-up and a top-down approach is that every gesture significantly modifies the running algorithms in bottom-up systems. By that, I mean that our gestures during a performance make significant, or effective, changes to the running algorithms[16]. The same does not necessarily apply when we type on a

[14]The sketch on Figure 3.4 is a simplified version of controllers like the Korg nanoPAD or the AKAI MPC series.

[15]See Figure 3.3 left-hand side for the GUI prototype.

[16]I will elaborate more on this aspect in the next chapter (Study B), where I present a framework for gestural interactions and highlight the distinction between bottom-up and

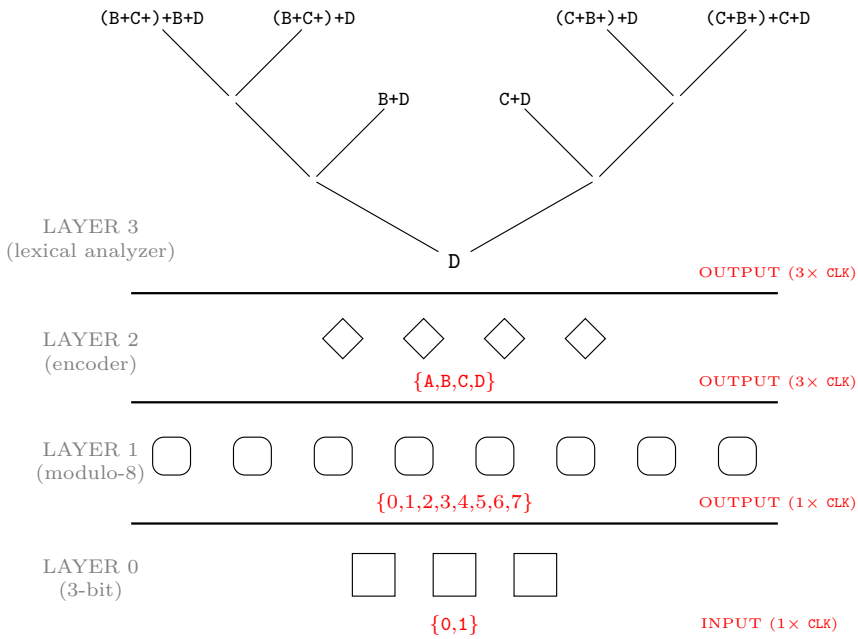


Figure 3.5: Diagrammatic description of the interface, as presented in [40], [42]. Layer 0 is the bottom-level where the user provides the input in bits (machine language). Layer 1 is the first level of computation, implementing a modulo-8 function (decimal representation of numbers, 0-7). Layer 2 represents the output of the encoder with four symbols. Layer 3 shows the lexical analysis where 7 tokens are generated. The sets of possible input/output values are shown within the curly brackets on each layer. On the right-hand side, the clock division of each layer is shown.

keyboard, where long sequences of individual characters are typed to form a programmable command. A series of individually typed characters in a text editor can be gesturally performed (or rendered) in many different manners, as we mistype a character, delete it, and so forth. Contrary, in bottom-up systems, every choice we make almost immediately affects the running algorithms. The specifics can differ between systems, but a similar logic applies for many bottom-up systems such as the stateLogic machine, Al-jazzari, Betablocker and the TOPLAP App.

However, Orca[17] (`https://100r.co/site/orca.html`), a two-dimensional tracker, is a system that also presents a characteristic that we may call *algorithmic significance* of gestures [35], meaning that all gestures we produce modify the running algorithms. But there are some significant differences. Interestingly, while Orca facilitates single-letter commands, thus, in a sense

top-down systems.

[17] `https://github.com/hundredrabbits/Orca`

facilitating direct manipulation, at the same time, the relatively large number of input commands (26 commands in total) does not facilitate recognition processes. Unlike Orca, the bottom-up systems presented above do not rely on single-letter commands and do not employ retrieval processes, also known as recall [18], but recognition processes instead. So, whereas Orca presents some characteristics that at first glance could be seen as a bottom-up system, at the same time it differs substantially from all other bottom-up systems, most notably in the fact the input commands are likely to engage retrieval processes.

3.4.2 Systems

On the system side, a bottom-up approach exemplifies the plasticity, or malleability, of the user interface. The interface between the coder and the system is dynamic, and the rules can change as we go. Maybe the best example to illustrate this in action is iMac Music, because it points directly to the craft. In all bottom-up systems, we construct the levels of abstraction progressively, and sometimes the rules can change dynamically. To elaborate, alternating an instruction set may be as easy as loading a new preset bank of instructions, or we can experiment on-the-fly one by one with new instructions, or even shuffle them (as in stack-based systems like Al-jazzari and TOPLAP App). But in a top-down system like the Sema ecosystem, changing the parsing rules in JavaScript can be challenging, necessitating that users possess a solid understanding of parsers, regular expressions, and related concepts.

In contrast, experimenting with grammatical rules in a bottom-up manner can be more accessible, as the users can easily test new grammars or instruction sets and evaluate their utility in a stream-based fashion. For instance, in Al-jazzari, each agent starts moving and generates music based on the instruction set commands, shown in an iconic preview and selected using a joypad. In that case, modifying an instruction set on the fly would immediately affect the agents' behavior, causing them to stop, clash, and so on. Such on-the-fly experimentation presents possibilities for learning and testing the system during practice without necessarily knowing in advance how to write the parsing rules.

The stateLogic machine demonstrates how to start live coding on the lowest level of information – that of individual bits – without having familiarity with the interface. In the first revision [42], in 2014, we added a lexical analyzer on top of the system to recognize regular expressions. The re-design of the interface [40], in 2022, enabled us to program the interface using a knob, which affords continuous control. To the best of my knowledge, only Approximate Programming by Kiefer [125], CodeKlavier CKalcuλator, and the redesign of the stateLogic machine offer continuous gestural interactions in live coding systems. The qualities of interaction in all three systems present diverse characteristics. For instance, in CodeKlavier CKalcuλator, the piano-coder performs sound-producing gestures to write the program. In the redesign of the stateLogic machine, we identified that different envelopes of gestures, either abrupt or with a steady pace, present distinct characteristics in the

[18] APA Dictionary of Psychology, two-process model of recall, `https://dictionary.apa.org/two-process-model-of-recall` (accessed: 2023-09-07)

behavior of the output. We conducted a statistical analysis which suggests that predictive models can be developed to inform the users based on their continuous gestural interactions [40]. While we did not implement a predictive algorithm in practice, the study opens a window to the possibility of predicting the generated tokens from continuous bodily movements. Such a prototype can be a ground-breaking contribution for live interfaces as it may tame the increased amounts of unpredictability induced by direct manipulation interfaces.

Another characteristic of all bottom-up systems is that none of them can crash during a performance, except the iMac Music which already presents a radical approach. It is interesting to notice that also Orca does not crash[19]. I consider this fact a weak design criterion for bottom-up systems. Failure is known to be highly appreciated within the community. If a system cannot crash, we endanger "playing" with the interfaces, whereas we should aim to *perform* with the interfaces. I will elaborate more on this issue in section 3.5, and at length in Study D.

3.4.3 Mapping

The design of the stateLogic machine demonstrates a *dynamic interface* in practice. The user controls the 3-bit input and on every clock cycle the system traverses a fully connected directed graph (Figure 3.6). This presents an input interface where its meaning is contextually updated based on the output [126]. It shares similarities with *dynamic mapping strategies* [127] but the interface is a directed graph, which makes it more akin to Kiefer's Approximate Programming [125], that uses a tree algorithm.

The case of dynamic interfaces, including dynamic mapping techniques, offers a new window to essentially transform our understanding of *gestural control* into *gestural interaction*. I try to make this distinction throughout this thesis, as the term "gestural interaction" is also part of the title of this manuscript. My main argument is that specific systems can always be controlled to a certain extent. This is certainly not a new understanding of mapping for electronic and computer music as it has been expressed in many different cases. What I observe and experience, and is in agreement with previous practices [73], [128], is that above this vague threshold of control, the only way to learn and interact with the interface is to explore the space of possibilities, which can be seen as an on-the-fly trial-and-error exercise for decision making. Similar to other interactive systems, this exploratory practice is informed by our mental model of the system's workings. In the case of the stateLogic machine, we can eventually learn certain patterns of interactivities. The difference with previous approaches for exploring a generative space [129] is that the user has to reach a goal using the stateLogic machine. In this case, the goal is to generate the desirable symbols or tokens from the initial input layer of individual bits. So, there is a distinct correspondence between input and output commands where the user steers the process and navigates a mapping strategy that is updated

[19]The author of Orca, Devine Lu Linvega, says at 33:55 that "I don't think you can crash Orca" `https://podcasts.apple.com/se/podcast/future-of-coding/id1265527976?i=1000467132034`

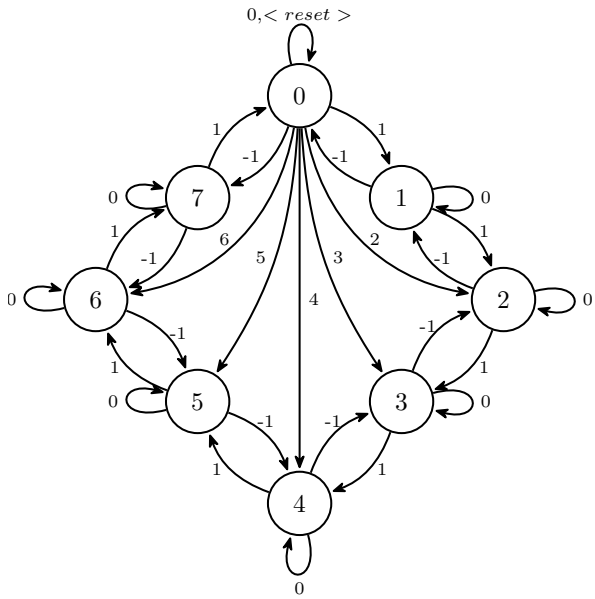


Figure 3.6: Fully-connected directed graph showing examples of transitions, representing the modulo-8 functionality and the workings of the modified 3-bit counter machine.

based on context.

3.4.4 A simple example of a musical improvisation system

Here, I describe with a simple example how the stateLogic machine equipped with the lexical analyzer, as presented in [42], would operate for an improvisation setting, similar to systems for which Dahlstedt introduces the term *systemic improvisation* [89]. The system was already described above in the paragraph "User interaction" (section 3.4.1) and a high-level representation is shown in Figure 3.5.

Let three performers (X, Y and Z) improvise in a turn-taking manner. The improvisation session has a time signature of 3/4. One out of three performers serves as a conductor, simply to facilitate the decisions of who is playing for the next measure. Each performer represents one of the 3-bit inputs given to the stateLogic machine. This is shown in Figure 3.5 as `Layer 0`. The binary representation from the least-significant bit (LS) to the most-significant bit (MS) in little endianness corresponds to `ZYX` binary representation for the performers, with `1` indicating the performer is playing and `0` indicating the performer is resting. During the current measure the conductor decides who is playing for the *next measure*. Layer 1, the output of the modulo-8 function, determines the diatonic mode for the next measure, by simply mapping the output to a diatonic mode (`0` is mapped to a whole rest).

A simple example is shown in Table 3.2. Let the performer X (LS bit)

begin the improvisation session and serve as the conductor. The current output of Layer 1 is set to zero (0), which corresponds to a whole rest, and serves as the initialization condition. This is tricky, as performer X is assigned to be active during the 1st measure, but is not playing during the 1st measure because the output of Layer 1 is set to zero. So, no performer is playing during the first measure and the current output of Layer 1 is mapped to a whole rest. For the 2nd measure, the output of Layer 1 will be mapped to *diatonic mode I* (the sum of the input and output of the previous measure, $0 + 1 = 1$, as there is a homomorphic mapping between the diatonic modes and the output of the modulo-8 function). During the 1st measure, the conductor decides that performers YX[20] will be playing during the 2nd measure. Thus, the diatonic mode for the 3rd measure would be IV and so on.

Table 3.2: Table showing the rules of the improvisation.

Meter	1st	2nd	3rd	4th	5th	6th
Performer	X	YX	ZYX	Z	X	-
Diatonic mode	Rest	I	IV	III	VII	Rest
Layer 1 (output)	0	1	4	3	7	0
Input (decimal)	1	3	7	4	1	0
Input (binary)	001	011	111	100	001	000

Table 3.2 shows on the output of Layer 1, but not the output of Layer 2 (symbols) and Layer 3 (tokens). The time signature is 3/4 to denote the `3x CLK` shown in Figure 3.5.

Table 3.2 shows only Layer 1 and it is the first layer that we apply a simple *binary operation* between the input and the current output[21]. Layer 2, is the first level of abstraction (Figure 3.5), where the output of the modulo-8 function is encoded in four symbols (`A`, `B`, `C`, `D`). Layer 3, is the second level of abstraction where the symbols (or bytes) are sent to a lexical analyzer that generates seven tokens (or words).

The time signature of the improvisation was set to *3/4* (Figure 3.7). This is because the output of Layer 1 is a 3-bit output in binary representation. This 3-bit output is fed to a decoding and encoding machine, that in response outputs four symbols (`A`, `B`, `C`, `D`). The decoder/encoder operates at 3×`CLK`, whereas the input/output layers (Layer 0, Layer 1 in Figure 3.5) operate at 1×`CLK`. Each of the symbols is generated on a *variable-length* of steps[22]. For instance, symbol `B` requires a two-step process that spans across two quarters, whereas `A` a single step. That practically means that the time signature was set to *3/4* so that we can apply another kind of musical instruction each time a new symbol is produced. So, when symbol `B` is produced then we can construct a binary tree with the possible tokens to be recognized by the system (i.e., `B+D, (B+C+)+D, (B+C+)+B+D`). For example, we may apply different kinds of

[20] Indicates the ZYX binary notation. So, if performer X and performer Y are active and performer Z inactive, Z is removed from the ZYX notation.

[21] The addition operation is in 2s complement, that is a modulo-8 function as shown in Figure 3.6. For an exact description see [41].

[22] For the specifics of the encoding process see previous work [42] or Paper III [40].

articulation for the generated symbols, say *staccato* for `A` and *legato* for `B`. Then, Layer 3 requires one or more symbols to generate a single token, and we can apply co-articulation on a larger time scale.

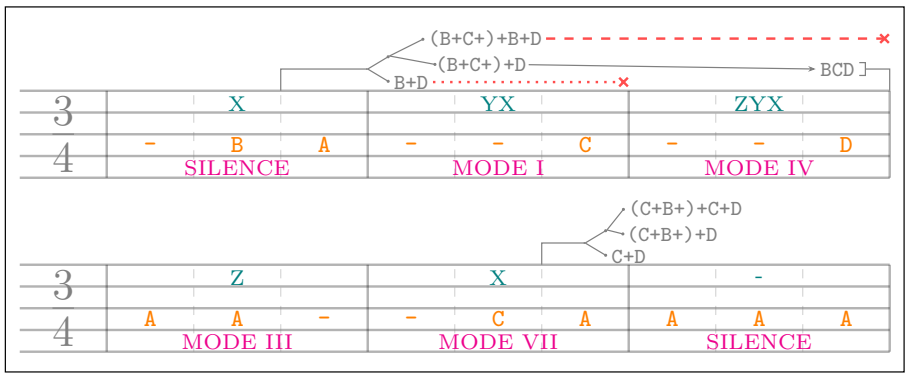


Figure 3.7: The score shows the transitions of Table 3.2. X, Y, and Z are the three performers. `A`, `B`, `C`, `D` are the symbols. During the first bar, a tree structure begins after symbol `B` is produced, indicating the possible token generation that starts with a `B` (i.e., `B+D`, `(B+C+)+D`, `(B+C+)+B+D`). At the end of the third bar, the token `(B+C+)+D` is recognized by the lexical analyzer. The generation of token `B+D` is excluded when symbol `C` is generated, as shown with a red dotted arrow.

For the prototype that affords continuous gestures presented in 2022 [40], the process is somewhat more entangled, and the above example would not elucidate the exact workings of the system, but there are a couple of things that would make immediate sense. Essentially, the conductor no longer directly decides who plays for the next meter. Instead, the conductor is gesturing, corresponding to some output from the encoder. It is then the encoder that assigns who is going to be playing for the next meter as part of a feedback loop between the 3-bit input to the counter and the output of the encoder (for a full description, see pseudocode in [40]).

3.5 Implications

There are interesting takeouts about how to use notation in bottom-up systems, mainly because, in many cases, the code updates are so fast that it can be hard for the users to follow them. The reason for this is related to the fact that such systems afford faster interactions, reducing the delayed auditory feedback. For instance, in Approximate Programming – which I do not categorize as a bottom-up system but it certainly shares some similarities – the code updates are so fast that it makes it impossible to read the prescriptive part of the code. In the redesign of the stateLogic machine the continuous control interface requires us to use faster clock updates for the interactive prototype. This transformed the output of the 3-bit counter (Layer 1 in Figure 3.5) to a more

descriptive representation as the current output of the 3-bit counter is displayed as piece-wise constant curves (Figure 3.3). Furthermore, the output of the encoder is visually suppressed and is implicitly shown to the next abstraction level of the generated tokens as the most probable code completion. Only the probable tokens are shown using prescriptive notation with code highlighting. This demonstrates in practice how to devise the abstraction levels in bottom-up systems in a manner that can be informative to the user based on cognitive constraints. In Figure 3.3, this is shown by displaying an increased size for the probable tokens and a reduced size for the excluded tokens. In this case, low-level abstractions are displayed using descriptive notation, whereas higher-level abstractions are displayed using prescriptive notation.

Continuous gestural control is a fairly unexplored area in musical live coding. Whereas a typical case of continuous gestures would involve parameter mapping using a mouse or more sophisticated DMIs like the Stenophone [130], in bottom-up systems we see how continuous bodily movement is used to write the computer programs. Whereas continuous control in HCI is known to be *not* accurate, in live coding it is fascinating to see how it brings about new opportunities that enable us to discuss gestural interactions. I will elaborate more on this topic in the Discussion (section 7).

I posit that risk is a crucial component of bottom-up systems. All bottom-up systems presented here are impervious to risk, meaning that these systems cannot crash during a performance. This trait provokes inquiries into such systems' creative potential, as failure is not a possible outcome. Failure can induce surprise, which is an important factor for creativity. Consequently, it becomes imperative to reassess how risk can be incorporated into these systems. Among the systems discussed, only Jonathan Reus's conceptual performance with the iMac Music system directly addresses risk in the most literal manner – as the computer circuits may melt or catch on fire under certain circumstances.

3.6 Publications in relation to Study A

Study A combines Paper I, II, III and VII, as shown in Table 1.1. The central contribution in Study A is Paper III [40] which introduces the term *bottom-up live coding*. It also extends previous work to demonstrate how continuous control can be used for a live coding interface, and it is an applied contribution. Our systems presented in this study have a listening component on the machine side. For instance, on top of the stateLogic machine, a lexical analyzer was added to listen to the symbolic output of the system. This symbolic listener differs from how Rowe [1] is listening and sensing the audio output and/or the acoustic environment.

Paper I, II, and VII mainly contribute to the conceptual understanding of the study and also introduce a methodological analysis of observations of similar systems. All three articles will be discussed in greater detail in Study B, where I focus on embodiment and gestures. Their role in Study A is to support the theoretical background of the newly introduced term *bottom-up live coding*, and briefly discuss from an embodied view how this term has an

effect on our gestural interactions. In brief, Study A focuses on the system-side whereas Study B on the human-side.

Chapter 4

Study B: Embodiment and musical gestures in machine musicianship

Embodiment, a concept from psychology and philosophy, posits that human cognition does not differentiate between mind, body, and brain, transcending the Cartesian division between mind and body. Within this context, this thesis sees that embodiment "assumes that subjective experiences are expressed in bodily changes" [30, p. 236]. The consequences of embodied cognition permeate numerous research areas, including human-computer interaction, interaction design, and musical interfaces.

Musical gestures are corporeal articulations of musical experiences and are seen as bodily expressions capable of unifying movement and meaning [131]. They are embodied percepts that can serve as intuitive conduits in music performance, given their relationship to expressive interactions [31]. Machine musicianship can incorporate designs that include musical gestures and focuses on investigating how computing technology extends our interactions with and understanding of music [1].

The previous conceptualization of musical gestures possesses a philosophical dimension, suggesting an affective and semantic connection between the human body and music. Embodiment, specifically the sensorimotor theory of embodied cognition, posits perception as a dynamic process [33] rather than a static representation of experience. This perspective transcends the traditional approaches to study perception and cognition, such as oversimplification of real-world situations, as typically used as "purified" stimuli in highly controlled experimental conditions that lack ecological validity. In contrast, an embodied perspective suggests that we interact with our surrounding environment and create our understanding by means of these interactions with it.

A question arises when contemplating whether music is only an evolutionary by-product, as posited by Pinker [132] in "How the Mind Works", or rather an adaptation in itself – an emergent social activity that evokes emotions like

joy, sadness, sorrow, nostalgia and awe. Whereas Pinker acknowledged the evolutionary benefits of music, as it is undoubtedly an activity that promotes cognitive and social skill development, it is really hard to answer whether one should take an adaptationist stance or a non-adaptationist [71]. Music is surely a socially emergent phenomenon that facilitates forming larger groups and enables individuals to collectively overcome challenges, but it is also possible that these skills were developed through a process of natural drift [33] – indicating a co-evolved process, one not fostering the survival of the best but the "good enough."

4.1 Enaction with musical interfaces

Action and perception are intertwined processes that influence one another. The stance I have adopted suggests an enacted theory of embodied cognition [32], [33], positing that representations are not requisite for our understanding[1]. This enacted perspective asserts that sensorimotor prediction is the main mechanism at the center of the sensorimotor theory of embodied cognition. We act, using some corporeal articulations, and we expect some specific outcomes. Typically, when using interfaces for music performance we perform hand and full-body gestures in a manner that facilitates our expectations. Thus, we perform gestural control with roughly foreseen outcomes.

Typically we design gestural control for musical interfaces with the intention to afford immediacy of action, and promote naturalness of the given interaction. For example, when designing an air-instrument [53], such as a theremin, to manipulate the pitch of a sound generator, it feels more intuitive to hear the pitch ascend when raising one's hands and descend when lowering them. Designs that facilitate this perceived naturalness of the interface can be explained through the concept of embodied metaphors [94]. Metaphors are extensively used for musical control and transformation of musical material [10]. Recognizing the delicate balance between intuition and conscious mental effort is desirable in musical improvisation [56]. The challenge lies in identifying this aspect in live coding and determining how to design live coding interfaces that minimize the distance between logical actions and intuition. This balance is not straightforward, as live coding interfaces are dynamically modified in time, which can lead the musician to a position wherein they have little to no control over the interface. For instance, it is often the case that we have a rough understanding of all the ongoing processes, and this state of affairs may eventually lead to a system crash.

Machine musicianship provides intuitive interfaces that can considerably reduce the cognitive demands and effort required from performers during the interaction, as it focuses on "emulating human musical understanding" [2, chap. 1]. In traditional music performance, sensorimotor prediction is a core mechanism that musicians use during performance [86]. The same logic more or less applies to most of the musical interfaces developed for music performance,

[1]Typically, a musical interface heavily relies on some sort of rigid representation, although we will see that this is not exactly the case in live coding systems.

especially when we discuss gestural control with a musical interface. Conversely, live coding lacks immediate sensorimotor prediction, as anticipating the inner workings of an algorithmic process is not straightforward. Of course, there is some understanding by the performer of what is to be expected from a new code evaluation. Still, the generativity of code usually induces substantial mental effort to the performers. The excessive mental effort may impede the pace and flow of their performance while potentially garnering appreciation from the audience. While most live coders use the keyboard to interface with the programming language, there are various other manners of gestural interaction during a live coding performance, and various artists adopt diverse interpretations and approaches.

4.2 Embodiment and gesture in live coding systems and practices

While gestural control is slowly garnering increasing interest in live coding practices, it remains a relatively unexplored area within musical live coding. Various live coding expressive styles exist, such as live coding choreography [133], wherein embodiment and gestural expressions are essential components. However, similar conventional practices are not yet evident for musicians. The recent inception of the annual online workshop Hybrid Live Coding Interfaces[2] has sparked widespread interest, which can boost the interest in bodily movement.

The appreciation of various notions of embodiment is well acknowledged within the musicians' community, as observed during the NIME 2018 workshop [134]. With such varied interpretations of embodiment, there is also a wide-ranging focus on gestural interfaces. Cases that will be explored here include Code LiveCode Live and Gewording[3] by Baalman [122], [135], Approximate Programming by Kiefer [125], CodeKlavier by Noriega and Veinberg [136], iMac Music by Reus [113], stateLogic machine by Diapoulis [41], Betablocker, and Al-jazzari by Griffiths [112], and Threnoscope by Magnusson [137]. These particular systems were selected because they represent diverse cases of how gestures are used in live coding. Whereas I have already conducted video observations on several articles that extend beyond the scope of this study, this section aims to complement the relevant articles [35]–[37] by emphasizing the value of recognition and retrieval processes in gestural interactions.

4.2.1 Performance systems: Musical gestures in practice

Baalman has been developing a gestural interface for live coding, called Gewording, that enables the performer to program the machine using full bodily movements. While the system is still under development, Baalman reports [135] that the system blurs the line between "design time" and "performance time", a concept discussed by Nilsson [138] as two distinct processes, one being out of

[2] `https://hybrid-livecode.pubpub.org/`
[3] `https://marijebaalman.eu/projects/wezen-gewording.html`

time while the other unfolds in real-time. Essentially, Baalman modifies the written program during a performance by typing on her laptop and immediately testing the new space of possibilities with the gestural interface. I would guess that the specifics of Baalman's system are not reported simply because it is an ongoing exploration of how sound and gestures can correspond with each other. However, anecdotal evidence from Baalman suggests that the performer must remember gestural sequences corresponding to specific programming commands. It is unclear whether this *gestural vocabulary* applies any algorithmic modifications to the running program or simply controls different parameter mappings. Anecdotal evidence by Baalman suggests that the cognitive effort required of the performer is highly demanding, necessitating attentiveness to a gestural vocabulary of serial skilled actions that the gestural recognition algorithm will interpret during a performance.

In another performance system, Code LiveCode Live [122], Baalman demonstrates the utilization of physical input data [54] as musical material. This system has redefined the notion of musical gestures in live coding, exemplifying how typing can be both observable and significant [37]. The system uses the sounds of typing itself as raw musical material and incorporates additional built-in sensors from the laptop, such as the trackpad and CPU temperature, among others.

Kiefer [125] developed a gesturally controlled system called Approximate Programming (AP), which relies on the real-time rearrangement of binary trees that form the sound synthesis engine. The algorithm is based on evolutionary computations [127] and exhibits chaotic behavior. Anecdotal evidence from Kiefer[4] indicates that the system's behavior can be completely unpredictable, leading to surprises for the live coder during a performance. Essentially, the generative space of such systems can be vast, requiring the performer to explore them non-linearly. Here, the performer explores the generative space by responding pre-reflectively using gestures without conscious awareness of their implications, making it difficult for the performer to follow the code updates on the prescriptive part of the written program.

CodeKlavier CKalcuλator [136], one of CodeKlavier's module, by Noriega and Veinberg represents a significant contribution that pushes the boundaries of live coding. This approach shifts the usage from standard live coding practices, where the composer-programmer interacts with a computer keyboard, to a practice of interfacing with a piano keyboard. It is one of the few instances where a traditional musical instrument is used as an interface to live code. CodeKlavier CKalkulator recognizes musical patterns using MIDI data to write the computer program. In this case, the coder has to recall the melodic sequences corresponding to the desired command to carry out the arithmetic operations. Here, the piano-coder employs retrieval processes which may increase the cognitive effort necessary to affect the performance.

Reus presented iMac Music [113] wherein he gesturally interacts with the internal wiring of a personal computer to live code audio-visual outcomes. This radical approach to live coding demonstrates how the coder can change the

[4]Kiefer's presentation of AP in ICLC 2015 `https://youtu.be/WwhpRtxq1Kg?t=3417`

running algorithmic processes within a computer by gesturally intervening with the hardware components themselves. Here, Reus demonstrates how to gesture *inside the hardware*, but at the same time, is *above the hardware*, inspecting how to manipulate the system. This panoptic position of the performer suggests that he can act quickly based on recognition processes.

The stateLogic machine is a live coding system that interfaces at the lowest level of information, that of individual bits. Here, the notion of tangible bits [139] is expressed in its literal manifestation. As in iMac Music, this musical interface facilitates recognition rather than retrieval. The initial design of the interface was developed as an approach to *live hardware coding* and was implemented on solderless breadboards using transistor-transistor logic (TTL) integrated circuits (ICs). In the initial design, three input buttons initiated operations; after redesign of the interface in 2022, a single knob was used for input control. The system employs stream-based code updates, and the levels of abstraction are progressively built on-the-fly, requiring the user to be adapt at gestural action. The level of control over the interface depends on the period of the stream-based updates, with clock updates faster than 0.5 seconds becoming a challenging endeavor. This is the case in the redesign, as the code updates faster than 100 milliseconds, making it hard to claim control over the interface. Furthermore, as the control over the interface depends on the various levels of abstraction, different cognitive mechanisms are required to control the first level of abstraction (see Layer 1 in Figure 3.5), and the higher levels of abstraction (see Layer 2 and Layer 3 in Figure 3.5). This is because a token will be generated after several clock cycles, meaning it may even require the employment of long-term memory processes (i.e., time intervals larger than 15 seconds).

The Threnoscope [137] is a live coding system incorporating prescriptive and descriptive notation. The descriptive aspect, which showcases structural and functional representations of the written code, allows users to adjust musical parameters related to the timbre of the resulting musical output. The system's primary objective is to establish and highlight structural relationships between prescriptive notation and their corresponding musical structures, achieved through the utilization of descriptive notation. The gestural control of the descriptive part of the notation is based on direct manipulation using the mouse, which facilitates recognition processes.

4.2.2 Recognition and retrieval with gestural interactions

A critical insight derived from examining various performance systems is recognizing the significance of cognitive constraints, as informed by HCI and psychological research on recognition and retrieval. Typically, a gestural interface promoting recognition affords direct manipulation (DM), whereas an interface necessitating the retrieval of gestural sequences is contingent upon the *complexity of the interface* (CoI). Notably, there is no obvious pattern among different systems regarding their facilitation of interactions that prioritize recognition over retrieval or vice versa. The relative level of CoI and DM varies on each system. The only observation one can assert is that most bottom-up

systems rely on recognition processes. Interestingly, Orca, an interface that affords single-key commands, thus employing direct manipulation, does not rely on recognition but requires retrieval processes. Although it is possible that a relatively large but limited number of commands can be mastered when the user has enough familiarization with the interface.

Baalman's Gewording also relies on retrieving gestural sequences, as performers must learn the gestural vocabulary, which can be a cognitively demanding task. In such cases, there are different requirements for the performed gestural vocabulary. The performer must recall the various elements of the vocabulary, and their corresponding spatiotemporal trajectories. Thus the cognitive mechanism of retrieval is two-fold, relying on both a discrete number of elements from the gestural vocabulary, as well as the spatiotemporal repertoire of gestures. Maybe a similar mechanism is at play in CodeKlavier CKalkulator (CC) although the difference is that in CC the performer can automate cognitive processes when playing the piano, as these are already learned sequential actions. Also, in CC the exact spatiotemporality of the gestural trajectories, including ancillary gestures that support the sound-producing gestures, do not have an effect on the running system, as the recognition process takes place on MIDI data. The similarity of CC with Gewording is that for both systems the performer relies on a gestural vocabulary that is retrieved from memory.

In contrast, Kiefer's Approximate Programming facilitates recognition processes, even though he acknowledges the absence of sensorimotor prediction. This can make learning the interface difficult, as we cannot anticipate how the system will behave on our input. The stateLogic machine affords recognition, and the user can predict how the system will behave on the input, although the redesign in 2022 introduces a stochastic component, the user therefore cannot accurately control the output. This stochastic component generates a different number of tokens based on the quality of the performed gestures, whether consisting of slow and steady movements or fast and abrupt movements. Such different qualities of gestures result in noticeable variations in the statistical distributions of the generated tokens. A plausible justification for this stochasticity is that direct manipulation with continuous control is less accurate than a push-button interface. Finally, Threnoscope affords recognition processes that are implemented in the visual interface by an affordance for the adjustment of the size of visualization rings (Figure 4.3).

4.3 How embodiment is expressed in live coding

Embodiment is essential in music performance [96] and in traditional performances musicians' bodily movements are expressed through overt gestures. These gestures provide expressive cues that the audience can perceive [140]. Performers can embody different *performance variations*, such as playing with high or low expressivity, which is discernible to the audience through bodily movements. This connection reveals that emotional content is closely intertwined with our sensorimotor network, suggesting that our musical experiences can be observed rather than induced by design; this would be a reductionist

approach [25, p.246].

Embodiment in live coding diverges significantly from traditional music performance. Collins argues that: "If we define music as requiring a certain sensorimotor engagement, live coding can be excluded [...]" [54, p. 114]. He also argues that there is an equivalence between motor skill acquisition and cognitive skill acquisition and that live coding could encompass different types of attention-oriented skills. There is also a view that sees live coding as a musical practice that suppresses motoric percepts in favour of high-level cognitive resources [141], [142]. I argue that embodiment in live coding is far more complex than traditional musicianship and the views above ignore how various musical activities (i.e., musical imagery, music listening and music-making) contribute to embodied percepts[37].

Salazar [143] questions whether a new definition of musical gestures is needed to explain live coding practices. Practitioners generally agree on the presence of embodiment in live coding performance, although the perspectives on how it is experienced vary widely. For example, Baalman [122] posits that even programming languages can influence our motor patterns, whereas Hutchins [144] does not experience embodied percepts when live coding on a keyboard, instead seeking embodiment in tactile interactions with sound synthesizers.

4.3.1 Gestural interaction and musical gestures

In this study, I will not endeavor to create a new definition of the gestures encompassing the numerous aspects of live coding. Such an effort would be substantial, as the literature on musical gestures spans from sonic gestures to bodily gestures [145]. Instead, I will explore what can be gleaned from observing bodily gestures in live coding, employing the current terminology of musical gestures as presented by Jensenius and colleagues [131]. Jensenius contends that gestures convey "meaning" through action [53, p. 65], while musical gestures involve the interplay between a "sign" and a "signifier" [53, p. 67]. Musical gestures are defined as "the combination of sound and motion" [53, p. 68]. Thus, bodily motion communicates meaning.

Throughout the thesis, I use the term "gestural interactions." The current agreed-upon term to denote control over a device for interactive music systems and DMIs is "gestural control" . Live coding is known to be overtly generative, making it hard to claim control in the first place. When interacting gesturally with the input interface, some uncertainty is infused into the performance, either because of the difficulty of the task on the human-side or because of the system's agency. The term "gestural interaction" denotes that the user is interacting with the system in an enactive manner and cannot necessarily claim control over it. This can be particularly the case in bottom-up systems, most especially when the code update rates are fast (i.e., less than 100ms).

4.3.2 The role of pre-reflective processes

Pre-reflective processes reflect a phenomenological approach to examining embodiment in music performance [31] and refer essentially to fast, automatic, and subconscious processes [146]. But programming is typically a slow and laborious process, not in the sense of requiring physical effort to play traditional musical instrument, but certainly requiring increased amounts of attention. Expression in traditional musical performance is associated with increasing amounts of physical effort, as this is reflected in the overt bodily movement of musicians, which results in increasing amounts of arousal. A typical example of this is when musicians perform with a visible amount of bodily sway, which is perceived as increased expressivity by audiences [140]. A musical live coding performance does not tend to show any overt bodily motion, likely due to the absence of sound-producing gestures. As this arousal component is missing during live coding we have to look for fast gestural interactions with the interface, which may activate sensorimotor synchronization (SMS) processes. Other manners to search for expressive cues can be investigated in musical structure, but I think this can be a really challenging endeavor given the generativity of live coding systems. A research project aiming to investigate expressivity in musical structure should definitely investigate aspects of live writing and how the writing process relates to it.

4.4 Contributions

The contributions presented here build on a theoretical perspective of embodied music cognition and sensorimotor theories of perception [31]. I start by presenting a conceptual framework that examines the characteristics of live coding systems based on gestural interactions (Paper I and Paper II). The outcome of this framework suggests that the constructed conceptual space is validated based on a cognitive paradigm where low-level and high-level programming constructs meet with the concreteness and the abstractness of the conceptual space, respectively. I then continue with an explanation of bodily gestures in *bottom-up systems* and discuss specific examples where pre-reflective processes are evident during live coding (Paper VII).

4.4.1 A framework for live coding systems on gestural interactions

A high-level description of musical live coding was introduced in Paper I [35]. At the bottom of Figure 4.1 there is a three-fold high-level explanation of live coding as *practice*, *agent* and *system*. A *system* in live coding seems to be necessary, whether it is a notation, a computer program, or a mental model. *Practice* is the only way to learn how to live code, as there is no curriculum in live coding. The *agent* in the middle represents both human- and machine-agents, as both co-create the musical, and sometimes coding, outcome. This high-level representation is coupled with a theoretical background informed by

both the study of musical perception and other aspects of music psychology [35], [36].

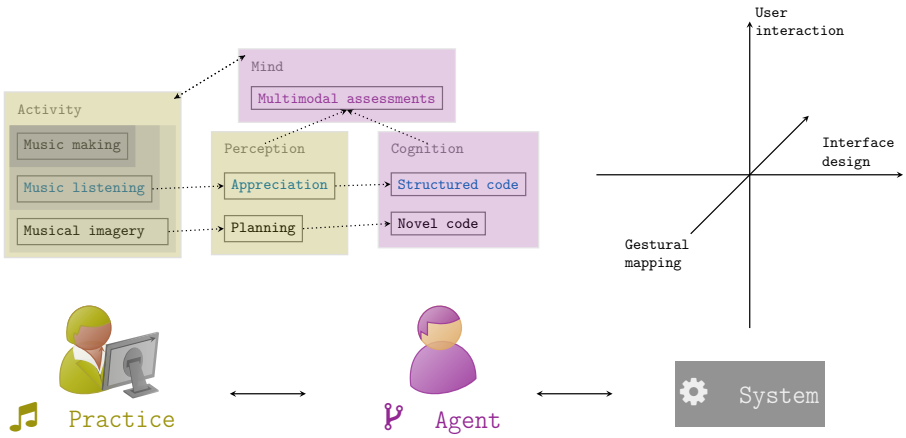


Figure 4.1: High-level description of musical live coding, as presented in Paper I [35].

The analytical framework we introduced examines how various performance systems differ from each other when observing or inferring the coder's bodily gestures. To facilitate communication of the framework, we decided to constrain it to a three-dimensional framework, harnessing the intuitive power of visualizations. The three dimensions considered are interface design (ID), gestural mapping (GM), and user interaction (UI). A fourth binary condition denotes whether the system has a "code-first" or "music-first" design [147]. The ID category places performance systems in a scheme similar to that used for digital musical instruments (DMIs), which views metaphorical designs as interfaces that use concepts from traditional musical instruments. For example, the guitar in the Guitar Hero video game is a DMI with a metaphorical design. In this framework, *metaphorical design* stands for interfaces that "hide" the programming interface from the user, whereas *literal design* stands for interfaces that expose the programming interface to the user. The second dimension (GM) examines whether the performed gestures impact the running algorithm. The third dimension addresses whether the interface uses *direct manipulation* to facilitate recognition processes or involves *algorithmic complexity*, which facilitates retrieval processes. The directionality of every dimension ranges from low-level concepts (concrete) to high-level concepts (abstract). Interestingly the results of the study showed that the computational design of the systems matches our cognitive paradigm of low- to high-level concepts. Thus, the low-level systems, like Al-jazzari and the stateLogic machine, are mapped on low-level concepts (see Griffiths and Diapoulis on Figure 4.2).

The framework is not meant to be exhaustive but aims to open a discussion in the live coding community on how we gesturally interact with our systems.

The novelty of the framework is its expressive power of visualizations, where multiple systems can be represented at once and show an overall conception of some basic system characteristics. There is no "best" way to construct a conceptual framework, and previous work on DMIs sometimes employs Venn-like diagrams [148], [149], spider plots [13] and other graphic conventions.

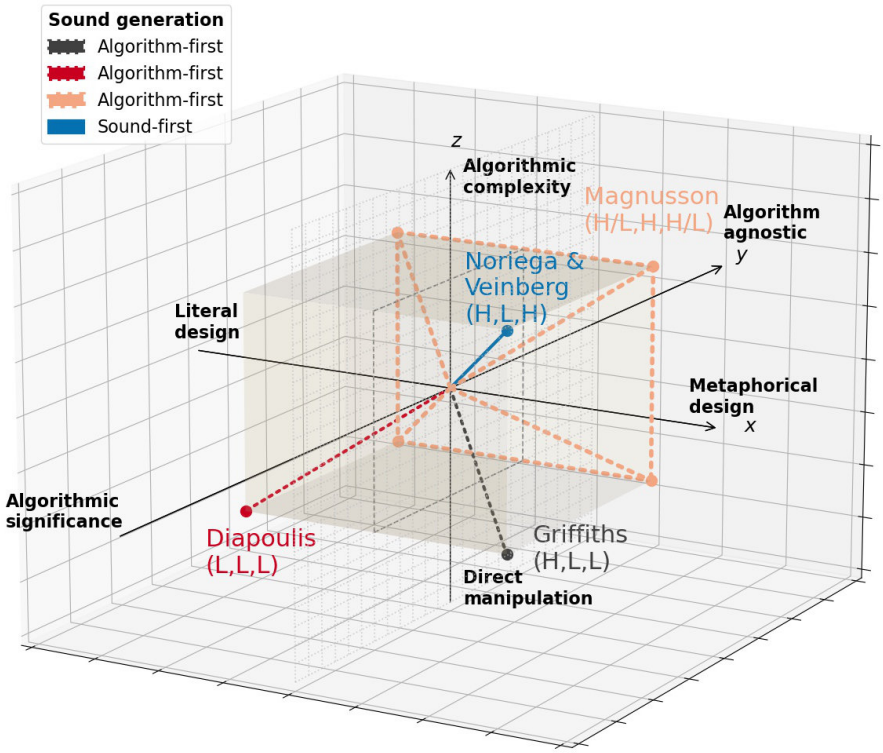


Figure 4.2: Analytical framework. X-axis: Interface Design (ID), Y-axis: Gestural Mapping (GM), Z-axis: User Interaction (UI). The framework should seen more similar to a *binary cube representation* than to a Cartesian plot. The Cartesian space may be misleading in this respect, but the aim is to represent existence or non-existence of different categories. The transparent plane, parallel to Y-axis, separates literal and metaphorical design spaces. Uppercase letters "H" and "L" code the high-level and low-level concepts, respectively.

At the time of writing the articles, I did not know some previous work on gestures and live coding. Sometimes it can be hard to find a corresponding article or video for a system, and sometimes also, the information provided by the authors is incomplete. Here, I discuss more systems with the aim of further testing the applicability of the framework by introducing two new examples (Approximate Programming and Gewording). I will discuss three cases: Threnoscope by Magnusson, Approximate Programming by Kiefer,

and Gewording by Baalman. I discuss Threnoscope here to visually map the system on the framework as this system stretches the expressive capacity of the framework. Threnoscope was discussed in Paper I, but was not visualized in the conceptual space. Furthermore, I would like to strengthen my argument about the distinction between GM and UI dimensions. The coding theme presented in Paper I (see Table 3, p. 7 in [35]) demonstrates that Threnoscope exceeds the expressive capacity of the framework by traversing the categories of *literal design* and *metaphorical design*, and the categories of *direct manipulation* and *algorithmic complexity*. On the other hand, for the dimension of GM, the system can be categorized as *algorithm agnostic*[5]. Visualizing Threnoscope in the framework will occupy a volume in the three-dimensional space, as shown in Figure 4.2.

Threnoscope divides the design of the interface into the prescriptive notation (code editor) and the descriptive notation (drone visualization), as shown in Figure 4.3. The prescriptive part affords the complexity of the interface, where the user has to type a sequence of individual characters for a single code evaluation. The descriptive part affords direct manipulation, using the mouse to modify parameters like the fundamental frequency of a drone sound. For both cases (prescriptive and descriptive parts), the user does not significantly modify the running algorithms using gestures (*algorithmic significance* on the GM dimension). By *significant modifications*, I mean the users can perform as many gestures as they like; this would not impact the running algorithms, but only adjust its parameters in the case of the descriptive part of the notation. In the prescriptive part of Threnoscope, the running algorithms would remain, unless we modify the code.

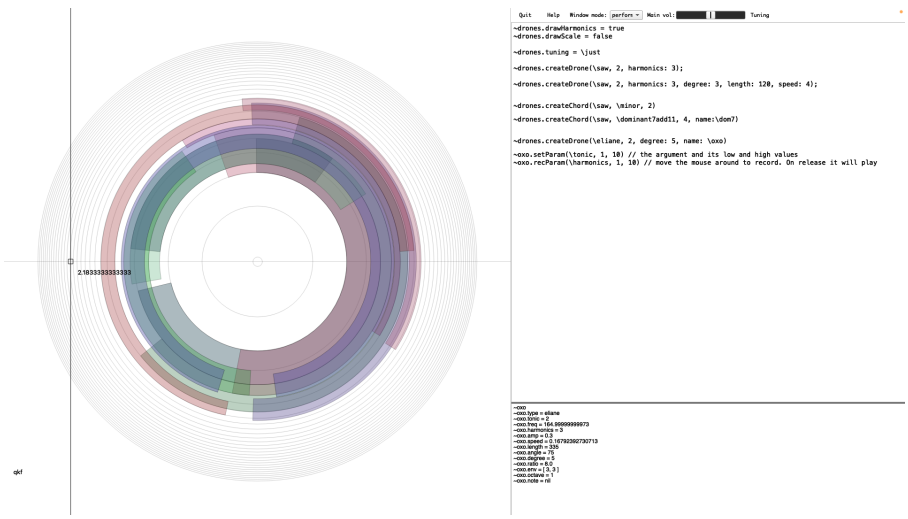


Figure 4.3: A screenshot of the graphical user interface and the text editor of the Threnoscope.

[5]The term *algorithm agnostic* I introduce, essentially describes that the performed gestures are *indifferent* to the running algorithms.

Typing a single line of code in the prescriptive part of the notation can be performed in numerous ways in terms of sensorimotor control. Unless it is causing execution of a code chunk, the user's gestures would not influence the system. Thus, whatever happens, typing a single line of code does not change the workings of the system. In the descriptive part of the notation, the user can control various parameters, which again does not mean that the underlying algorithms are modified on-the-fly. This is the main reason that the Threnoscope traverses bidirectionally both the ID and GM axes of the framework but not the UI axis (Figure 4.2).

Here, the term *algorithmic complexity* may be better expressed as a combination of the *complexity of the interface* and its *closeness to mapping*. In the case of Threnoscope, the interface affords both direct manipulation (DM) and increased complexity. But there is no way for the user to gesturally intervene in the running program. This is the case for any type of canonical live coding that does not incorporate any live writing rules or other bottom-up methodologies.

As discussed above, Gewording requires retrieval processes for the performer, since the performer has to recall elements from the corresponding gestural vocabulary to control the system. I am unclear whether the gestures in this case apply algorithmic modifications, because there is no analytical description of the system by Baalman. If the gestures can modify the running algorithms, then the system would occupy the whole three-dimensional space making the framework completely uninformative. If not, Gewording would occupy the same space on the framework as Threnoscope. This is because when Baalman is typing on the keyboard, the gestures are not linked to the running algorithms (what Nilsson [138] calls "design time"), and when performing, the gestures would apply significant modifications on the running algorithms ("performance time").

Approximate Programming is a system that employs a gestural controller to live code using evolutionary algorithms. The user can inspect the code updates on the prescriptive side of the notation, using either the mouse or a hardware MIDI controller. Thus, the first dimension (ID) can be seen as metaphorical design, as programming using sliders is not something we really do; we prefer to use sliders to control an interface, like a mixing desk. On the second dimension (GM), the system applies significant modifications to the running algorithms (algorithmic significance), and on the third dimension (UI), the system uses direct manipulation. Thus the system can be categorized in the same manner as Al-jazzari by Griffiths, maybe with the "music-first" binary condition as the code has to be written in advanced to be ready for performance time.

The framework contributes a visual representation that can be used by practitioners and researchers when designing live systems that afford gestural control. The framework aims to be practical and guide the practitioner on some relations between the systems, such as whether a system is using a literal or a metaphorical interface design. Here, I did not addressed anything related to the musical aesthetics of the sound, which suggests an incomplete analysis of musical gestures with respect to Jensenius's above-mentioned definition (i.e., musical gestures consist of "the combination of sound and motion"). Instead, I only addressed aspects related to motor perception and musical interfaces.

4.4.2 Gestures in bottom-up live coding

Systems that afford a bottom-up approach to live coding, like the stateLogic machine, Al-jazzari, and CodeKlavier CKalcuλator, are *not* agnostic, or indifferent, to algorithmic modifications. The main reason is that bottom-up systems are building algorithmic abstractions on-the-fly. When a command-input is set on a bottom-up system, it is either evaluated on-the-fly or awaits the necessary computational processes to be carried out. Until any remaining processed are completed, those already set will not change, unless they are modified from scratch. In this manner, every gesture we make intervenes in the workings of the running algorithms, which is different from controlling one or more parameters of an algorithm. The difference lies in that when controlling a parameter, we essentially act on the "surface" of the running algorithm, but we do not access and modify on-the-fly its very workings. This is not the case when we live code from the bottom-up, as the running algorithm is modified during performance. So, every gesture we make has a corresponding consequence on the running program, and the extent of this influence varies between systems. In the framework above (Figure 4.2), this aspect is shown with the dimension *Gestural Mapping*.

Gestural control in relation to DMIs has been a long-standing topic for discussion. Wessel presented an enactive approach to gestural control which gives rise to a phenomenon he called "babbling" [128], a term borrowed from the literature on speech development in infants when learning to match motoric serial skills to heard sounds. Wessel discusses *babbling* in relation to generative algorithms for gestural control of DMIs. This term shares some similarities to what I call *gestural interactions*, although I see some differences that become apparent in bottom-up systems. Wessel approached the topic from the viewpoint of precise control in gestural interfaces and also discussed issues related to the technical latency response of the systems. In live coding things are somewhat different, as the responsiveness of the system is not the focus. We always use some algorithm, and we know the algorithm has discrete steps which may require time to compute[6]. Of course, one can say the steps of the algorithm can always be carried out quickly so that we do not feel the system's latency. I would argue, and I think most of the live coders would agree here, that if we do not monitor, or maybe steer, the algorithm, then the algorithm becomes a black box. The argument here is that if we live code using a black box, it is impossible to form a mental model of the running algorithm. Maybe the difference between what Wessel calls "babbling" and how we steer our algorithms with gestures is that in live coding there is always a goal. We aim to progressively complete several code tasks to complete a desired pattern. In essence, the process of gesturing in live coding has an implicit or explicit goal.

Thus, I argue that when we gesture in live coding we are trying to meet our expectations of the sound outcome, similar to the infants babbling sounds trying to match the motor movement of the lips, but with the difference

[6]This is demonstrateed in the demo of the stateLogic machine where for each command to take action we have to wait for the next positive edge clock – as the "CPU" clock in this case of the video demo is 0.5 seconds.

that the coder is not attentive to mouth-motoric variations, but rather to algorithmic variations, by being attentive to the written code. When the present cognitive constraints cannot compensate for the information flow, then the gestures of the live coder can look more similar to babbling. When the rate of code updates does not exceed our cognitive limitations, we gesturally interact with the interface in more of a flow state. I see here that the pace and flow during performance can be mediated by pre-reflective processes when gesturally interacting with the interface, which I discuss in the next section.

4.4.3 Pre-reflective processes in live coding

The pursuit of bottom-up systems, along with developing a theoretical background about gestures, has resulted in an overall building of a theory of gestural interactions. By conducting observations of systems and practices from online videos [37], I developed a theoretical approach to examine whether pre-reflective processes are evident during live coding. The importance of the study lies in my reflections on musical practice, and I argue that the focus of our attention should turn to bottom-up systems when examining fast processes during performance. This theoretical pursuit is based on a sensorimotor view of cognition, which sees action and perception as tightly coupled.

Pre-reflective processes are cognitive processes that do not require conscious awareness and exhibit automaticity. During live coding, whether bottom-up or top-down, the performer is bodily engaged in an activity consisting of serial skilled actions [86]. This practice shares similarities with traditional music performance, but one important difference is that in live coding, the performer operates on an *open-loop motor program*. This is similar to playing the theremin, as there is no tactile sensory feedback to inform the performer about any actions that are not conducted in the desired manner. Simply put, when we press a character on the keyboard, there is no informative feedback for error-correction, and we can only see on the screen if a character is mistyped after the fact.

It is hard to imagine how would it be to live code using our closed-loop motor program, as there is not such case. How would the pre-reflective processes change qualitatively? Sayer [150] suggests that it is less likely that live coders would be engaged in subliminal processes in comparison to instrumentalists. I would agree that this is certainly the case, but it is unfair to compare a live coder to a guitarist, as the open- and closed-loop motor programs exhibit fundamental differences and opportunities.

4.5 Implications

My aim for the analytical framework on gestural interactions was to explore how systems and practices differ by employing concepts from HCI, musical interfaces and music psychology and perception. Such frameworks should not be seen as solid and set in stone but rather as challenges for live coders to reflect on their decisions when designing systems and consider how to go beyond them.

An important guiding principle in this work is to present the work of other people so as to enrich the discussion. Naturally, one of my concerns throughout the thesis is to communicate various practices and systems with the aim of raising awareness of what is out there. The live coding community has reached a point where robust and powerful tools are available, and I think this is where we can reflect and try to think outside the box. The term "live coding" was coined in 2003 [22], and the following ten years were mostly concerned with how to support the community and its practices and encourage it to flourish. Twenty years later, we have reached a point where anyone can open a web browser and start live code with no need for frantic software installation and problem-solving discussions in forums. Today, the discussions in live coding forums is more focused on creative applications and implications rather than on technical problem solving. I think this is the time we should focus on diversifying our tools and practices to get the most out of the creative potential of live coding as an artistic practice. As mentioned above, the live coding community documents itself [57], and, I also believe, reflects on itself. So now it is crucial to be informed about the diverse range of practices and systems used. I think this is the most important matter that we all should really care about.

I asserted that bottom-up systems differ in how we gesturally interact with the systems in comparison to *canonical live coding*. In bottom-up systems, almost every gesture we make impacts the running algorithms, but this is not true when typing on a text editor. It also seems that the interactivities I discuss in bottom-up systems differ from previous work on interactive music systems as was described by various authors. It may be that the difference relies on the goal-directedness one has during live coding, and this can be abstractly described as making an algorithmic modification in the running process.

Finally, live coding is identified as a practice that relies on the open-loop motor program, which excludes any informed sensory feedback from the system to the user while executing bodily gestures. In 2023, a hardware sound synthesizer equipped with motorized knobs was released[7,8]. Yet, the product does not afford any advanced programming that allows the users to co-perform using their closed-loop motor program other than a "coarse tuning mode", where the user slightly feels some tactile feedback because of the predefined tuning positions[9]. I think it is a matter of time until more creative applications will be developed around industrial products that afford actuated tactile feedback, as research on force-feedback has been already presented at NIME [151]. Thus, the implications of engaging our closed-loop motor program also extend to market opportunities.

[7] Nina, that synth with the motorised knobs, is finally heading your way - almost `https://www.musicradar.com/news/melbourne-nina-synth` (accessed 2023-08-25).

[8] Melbourne instruments `https://www.melbourneinstruments.com/` (accessed 2023-08-25).

[9] Motorized synth knobs: Gimmick or Gamechanger? // NINA by Melbourne Instruments Review and tutorial, `https://youtu.be/_9b1nVeQFJo?si=1-a_cJL4sWIB-zSl&t=263`

4.6 Publications in relation to Study B

Table 1.1 shows how the publications contribute to Study B. The most important contribution is from Paper VII [37], which essentially builds upon previous knowledge from Paper I and Paper II, although there are significant differences in the methodology, motivation and research questions. Contrarily, Paper I and Paper II are tightly related as Paper I is an extended version of Paper II. In Study B, I extend the examples put into the conceptual framework to examine the framework's applicability. Paper VII contributes a sensorimotor theoretical perspective in live coding that reflects on musical practice. All three related articles contribute conceptually and methodologically to examining gesture and embodiment in live coding.

Chapter 5

Study C: Creativity support technologies for live coding

Creativity is defined as the "ability to bring about new ideas" [56, p. 12]. In music psychology and perception studies, creativity is mostly related to novelty, which can be linked to any irregularities in the musical structure, harmonic, timbral and temporal variations. In computer arts, creativity also addresses issues of authorship and agency [89], [152], [153]. As discussed above, electronic music systems depend on a certain level of autonomy, which obscures the boundary between the actions of the system and those of the performer. A common method for modeling autonomy involves the use of agent-based systems. These systems, linked to creativity, present models that raise questions about matters of authorship and agency.

I start by presenting some historical accounts of creativity in interactive music systems and their use in musical improvisation. My aim is to present to the reader a brief timeline of interactive agent-based systems and then continue on to the specifics of creative technologies in live coding.

5.1 Interactive music systems and machine agents in improvisation

The research on interactive music systems and musical agents in machine improvisation has several decades of related work [2], [76], [154]–[156]. The first systems appeared in the 80s as the work on AI by Marvin Minsky [157], one of the first computer scientists who worked on agent-based systems. He introduced the idea of "frames", which are seen as precursors of object-oriented programming (OOP) and were conceptualized as *super-agents* [158]. Following this work, Robert Rowe presented his system *Cypher* [159] which has a listener and a performer. The system operates on symbolic data (MIDI), and several components were developed to analyze timbral, structural, and rhythmical characteristics. During the same period, George Lewis started experimenting

with his *Voyager* system, a dyadic performance system that he still uses today. Voyager is a music system that exploits self-organized patterns of symbolic data (MIDI) to accompany the performer. Lewis explains his cultural influences, as a black American, during the development of the system, where effort was put into non-hierarchical and flat structures. Around the 2000s, in France, Assayag and colleagues [155], [160] were experimenting with *OMax* system, and Pachet with the *Continuator* [161]. Both were influential systems with sophisticated algorithms that still have a major influence to the present day. These systems incorporate Markov models that operate on different levels and enable the performer to have complex machine listeners that can be linked to different structural levels of the composition/improvisation. After the 2000s, novel algorithms for agent-based systems were presented. Indicatively, Dahlstedt was using evolutionary algorithms to explore timbral spaces [56], and Collins [76] was developing directed graphs of interacting agents using neural networks for beat electronic music and accompaniment in instrumental improvisation.

A taxonomy of improvisation interfaces was presented in 2018 [162]. The authors' focus was on how creativity is related to agency and aesthetics. They identified *bottom-up autonomy* and *top-down autonomy*. For instance, a bottom-up approach would be following a self-organization approach, whereas a top-down would follow a hierarchical design and consider notions such as intention, belief, and desire. A similar view on machine autonomy that extends beyond agent-based systems for improvisation was presented by Tatar and Pasquier [163], where bottom-up autonomy is seen as encoded or heuristic, and top-down autonomy as agent-synthesis.

In the next section I continue to examine agent-based systems specific to live coding. In all musical improvisation tasks, including live coding, agent-based systems are acting on real-time processes, which presents some unique characteristics related to the "liveness" of the systems.

5.2 From liveness and musical agents to machine learning ecosystems

At the live programming workshop in 2013, Tanimoto [43] presented a revised version of his hierarchy of liveness[1]. The first ICLC was two years later in 2015, and more and more studies emerged looking at creativity support technologies in live coding. A short review of the last ten years in the field of agent-based systems for live coding was presented by Xambó in 2021, where a conceptual framework analyzing the *learnability* and *social interactivity* of the systems was presented [164].

In this section, I will start by presenting how I see liveness in musical live coding and I will continue to musical agents. I will close this section with a brief presentation of two machine learning ecosystems that afford textual programming practices.

[1] `https://github.com/liveprogramming/2013`

5.2.1 On liveness

Liveness has been a concept discussed in a variety of disciplines, from software engineering [43], [100] and human-computer interaction [165] to musical aesthetics [87], [166], media studies [167] and anthropology studies [168]. In live coding music performance, liveness is an inherent quality [101]. In this chapter, I will elaborate on what liveness is, how it is applied in live coding systems, and how liveness exhibits different facets during a performance. I will then present a description of liveness seen as a three-fold blend of *technical liveness*, *perceived liveness* and *felt liveness*.

Liveness in the aesthetics of electronic music is an issue with various interpretations and understandings. Whereas Croft [166] takes the stance that the liveness of musical interfaces should adhere to mechanistic characteristics to facilitate audience and performers' aesthetic appreciation, Emmerson [87] sees two central relationships related to performer's gestures and audience appreciation. The gestures may either be (a) only in the mental model of the performer or (b) so experimental that they are unpredictable. Regardless of the different perspectives on liveness, many composers believe that liveness is a useful term in electronic music and assert that it is one of the mechanisms that can serve to mediate expressive interactions (e.g., Croft [166]). In musical aesthetics, the discussion focuses on the relationship between the performer, the musical interface and the audience. This explanation of liveness involves how different parties (i.e., performers, audiences) perceive the liveness of the performance or system used, so I call this *perceived liveness* from now on.

Interestingly, aspects of liveness are also discussed from a technical perspective [43], [100]. I will be referring to this aspect as *technical liveness*. Tanimoto introduced a hierarchy of liveness for programming environments in 1990, which was extended to define two additional levels of liveness in 2013. Essentially, such a notion of liveness categorizes how "live" a programming environment is. For instance, a system might be as simple as a static representation, such as a timetable (level-1 liveness, called "informative"). Level-2 liveness requires the system to be "informative and significant", for instance an executable flowchart. Level-3 liveness requires the system to be "informative, significant and responsive", such as including edit-triggered updates. Exhibiting stream-driven updates, such as in a live feed on social networks, is level-4 liveness, called "informative, significant, responsive and live." Level-5 and level-6 livenesses were introduced in 2013 to complement the existing hierarchy and can be seen as *advanced levels of liveness*. Level-5 liveness, also known as "tactically predictive", makes available code previews to the user, such as the autocomplete functionality of a text editor. Level-6 liveness, also known as "strategically predictive", attempts to infer what the user will be doing, as in GitHub Co-pilot. Tanimoto metaphorically conceives of the advanced levels of liveness as algorithms that run on "negative timescales" (lookahead) as was noted during his keynote presentation in ICLC[2]. Essentially, Tanimoto uses this metaphor of *negative timescales* to refer to the property that the system can see in advance of the user's actions, desires, and intentions.

[2]Tanimoto's keynote presentation in 2015 `https://youtu.be/4cJANuMiq18`

In the context of live coding, liveness is not only related to its technical manifestation. Although Tanimoto [43] addresses the similarity of control and feedback in both musical live coding and conventional programming, I think they differ. I suggest that it might best be discussed in the context of flow and pace, something that is also suggested in the recently published book *Live coding: A user's manual* where it is described as the "vitality of liveness" [25, p. 167]. When live coders discuss flow, their main concern is with the continuity between the written code and the musical outcome. In Roberts and Wakefield [104], some live coders reported that the technical aspect of liveness, such as future projections of code (code previews) induce cognitively demanding characteristics. Instead of exploiting aspects of the technical dimension of liveness, practitioners focus on balancing pace and flow during a performance. To do so, live coders employ several techniques to compensate for the excessive demand on cognitive resources, such as using keyboard macros [169], defining mini languages, or developing highly constrained interfaces. Thus, the live coders focus mainly on controlling technical aspects of liveness, and employ musical aspects to facilitate a felt continuity during a performance. In a sense, one modality (sound) compensates for another (code). I will be calling this *felt liveness*, which I see as a fundamental quality appearing in a live coding performance.

To summarize this section, perceived liveness is related to perceptual aspects of the environment and the audience. Emmerson approaches liveness through a *cognition through perception* lens; Tanimoto approaches the topic from a purely objective view of the technicalities of systems; and live coders approach it *through lived experience*.

5.2.2 On musical agents

In the last decade, various live coding systems have incorporated musical agents in their system design [156], partly due to the lower entry level of machine listening and machine learning technologies, compared to ten or more years ago. In the early 2000s, it was rather difficult to develop systems for musical improvisation that employed agent-based designs, as they required an expert understanding of both programming software and hardware equipment. Whereas machine listening and machine learning are not necessary and sufficient conditions for an agent-based system, sometimes these terms are used interchangeably. Essentially, an agent-based system demonstrates some level of autonomy, and ongoing responsive processes during performance exhibit autonomy to a certain extent.

Musical agents typically assume a co-creative context, where humans and virtual agents co-perform. In live coding, this can be done in different ways but the most widely used approach is that the virtual agent responds to its environment. This can be done in different manners: for instance, by constructing network architectures of interactive agents, or by simply inducing the perception that there is a responding agent in the system. Other approaches include having two (or more) agents that can both write code simultaneously in their text editors. This is a common practice for the case of human-

human collaborative sessions. Agents can have turn-taking roles [170], or can collaboratively edit the same document in a manner similar to the practice of collaborative editing with Google docs (e.g., `https://flok.cc/`).

Xambó [164] analysed 8 systems, some of which do not afford any learning capabilities – like Autocode by Magnusson[3]– and some of which do not afford social interactivity, like Cibo [171]. Both Autocode and Cibo, are live coding systems that can carry a performance autonomously without a human agent. Interestingly, Autocode's video does not have an audience, so the performance is speculative in its conception. Their main difference is that Cibo incorporates a deep learning algorithm trained on a large dataset of code examples, whereas Autocode does not afford any learning capabilities, only several interacting agents that respond to one another. Various other systems were presented in Xambó's study, like Flock by Knotts [172], Cacharpo [173], Betablocker and more. These works, along with mature MIR technologies [174], [175] and machine learning, enable the flourishing of performance ecosystems like Sema [176] and FluCoMa [177].

Recent trends, such as advances in neural audio technologies [178] indicate that this proliferation of assistive technologies is starting to find its way into interactive music systems, although it is an open question whether use of *black-box* sound synthesis will ever become a trend in live coding. This is because if the coder is unable to form a mental model linking the written code and the generated musical outcome, then it is unclear how they would be able to live code. It remains to be seen how live coders might incorporate such technologies into their systems. For instance, recent developments used a *temperature* parameter to experiment with the entropy of the generated code [179]. How and if similar technologies will be used for audio generation is yet to be seen, but it is likely happening as we speak.

5.2.3 On machine learning ecosystems

Creativity support technologies in live coding are becoming more accessible as we get beyond the period of machine learning ecosystems. Two large projects have appeared around 2020, Sema and FluCoMa. Sema is largely inactive today, whereas FluCoMa is a large project with different facets active, including even a podcast[4]. The two ecosystems have different target audiences, but both have as their main focus an interest in making machine listening and machine learning easily accessible to users of musical programming environments. Sema focuses on delivering a web tool where users can write their own programming languages by accessing parsing rules and applying machine listening and machine learning within the same programming environment. The system uses JavaScript libraries such as TensorFlow for machine and deep learning.

On the other hand, FluCoMa focuses on complementing existing programming environments in an accessible manner to apply machine learning as a third-party workflow. In that sense, Sema is focused on more expert users

[3]ixi lang autocode livecoding for no one at University of London's Institute of Musical Research, `https://youtu.be/jWvzCzR_tus`

[4]FluCoMa podcast: `https://learn.flucoma.org/podcast/`

and aims to deliver a fully-contained solution for live coding and AI. At the same time, Sema assumes its users are somewhat familiar with advanced programming concepts, such as parsing rules. In contrast, FluCoMa aims to take advantage of users' preferences and familiarity with their own pre-existing preferred compositional workflows. Thus FluCoMa supports a broad range of programming languages for music, such as PureData, SC3, and MAX/MSP, whereas in Sema the coders can write their own individual programming languages.

5.3 Contributions

My contributions with this study is threefold: I bring (i) visualization technologies to be used for creativity support in live coding practice and musical analysis, (ii) a conceptual framework for designing agent-based systems, and (iii) a technical contribution to the topic of liveness and bottom-up systems. The contributions (i) and (ii) overlap to some extend because the conceptual framework is a visual map that makes use of non-linear and modular visualizations. Furthermore, contributions (ii) and (iii) also overlap, as liveness is extensively discussed in both.

5.3.1 Visualization technologies for creativity support

5.3.1.1 Live coding practice with a visual helper and sound visualizations

I explored how a visual helper can be used in live coding practice [38]. It is interesting that there remain very few papers that discuss how we practice live coding. The study by Nick Collins [54] is one the first published articles that discusses the practice of live coding, and reflects on the learning outcomes. Collins conducted daily sessions with Fredrik Olofsson and that study provides an outline of how to do isolation exercises, connectivity exercises and finding the implications for the repertoire (section 2.4.1). Previously, Olofsson had documented[5] another month-long daily session from scratch, and provided SuperCollider source code from these sessions. Also, Sorensen and Brown [169] presented practical guidelines and technical solutions that have arisen in the practice of live coding.

I conducted daily practice sessions for one month, during October 2022, with the aim of identifying how multimodal experience influences flow, pace, and subjective appreciation of the musical outcome [38]. Visualizations of sound levels and the spectrum of the sound were central to the design of the study, along with a GUI helper showing a list of UGens to the user (Figure 5.1). A pilot study was conducted in August 2022 for three weeks, where I mainly tested different experimental setups and study designs. I kept daily reflective diaries after each session as well as keeping a reproducible archive of the sessions, using the `History` class in SC3. Every session was blank-slate, meaning no

[5] redFriklivecodepracticeAug2006, `https://swiki.hfbk-hamburg.de/MusicTechnology/818` (accessed: 2023-09-04)

prewritten code was available other than that of initializing the experimental setup (booting the sound engine and activating the archival functionality).

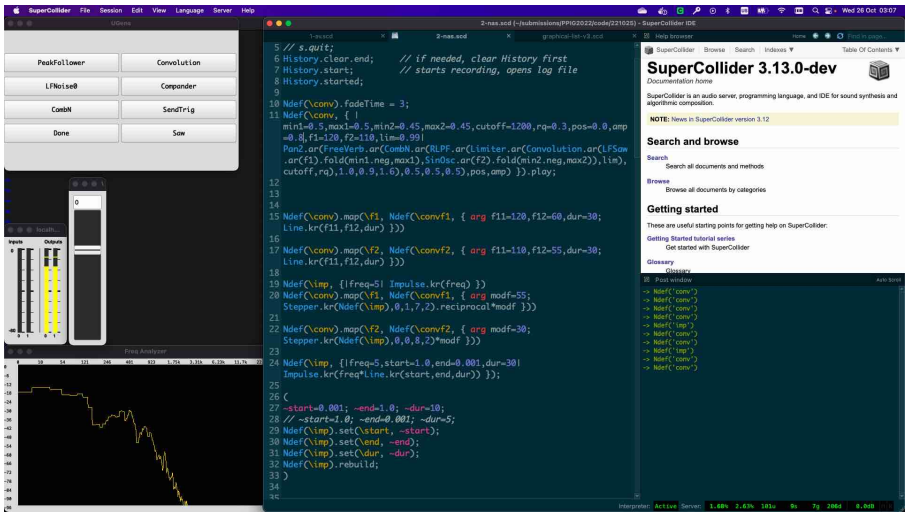


Figure 5.1: Experimental setup of the month-long study on practicing live coding. The sound levels and the spectrum are shown on the bottom left, the GUI helper on top left part of the screen.

The GUI helper was used to examine its influence during coding. A helper is not an agent. It is rather an automated program that is detached from the ongoing workflow of the coding session and instead seeks the attention of the practitioner. In the article, I discussed how such a tool could be evolved into an agent-based system by implementing some level of interactivity and potentially be used for program synthesis (a term to denote a functional code generation process). The helper showed that a limited number of optional actions could be useful to the performer, especially at the beginning of the sessions.

The study had two listening conditions – listening and non-listening to the sound outcome. Visualizations held a crucial role, especially during the non-listening session. Without any visual cues, I was unable to perceive whether any audio was produced at all. Certain actions are almost impossible using only our visual perception. For instance, it is rather hard to do transitions between sounds; it is difficult to understand rhythmical aspects of the sound; and it is also impossible to understand the density of the generated events.

The article contributes to code practices using assistive technologies, specifically to visualization technologies and the use of archival material in future developments. For instance, a large language model could be used to build a model on the archival material. That would definitely be a “personalized” model for coding, as the system has only “seen” a single user. Finally, the radical decision to turn off the audio output from the loudspeakers showed me that certain tasks become impossible, such as what Collins calls *connectivity exercises*, but also open a window for unforeseen ways to foster creativity while

not listening to the generated audio.

5.3.1.2 Circular representation of musical structure

In Diapoulis and Carlé 2023 [44], a circular representation of musical structure was constructed as part of a reproducible musical analysis on more than one hundred live coding performances (Figure 5.2). We suggest that such a visualization tool could be used both during practicing and performing live. The development of the tool is informed by music cognition and adheres to gestalt principles, such as good continuation, similarity, and proximity.

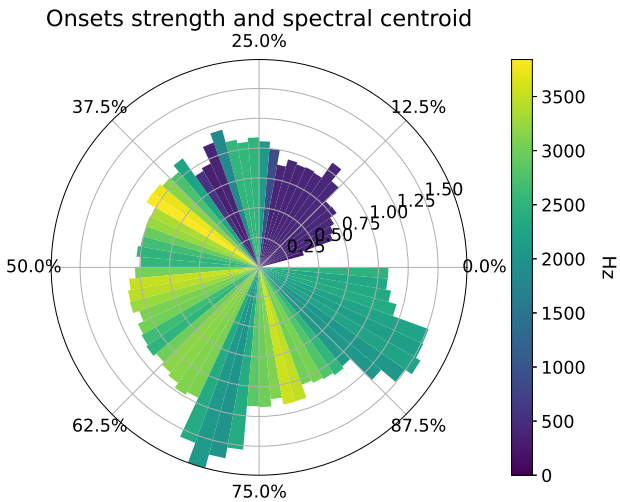


Figure 5.2: Circular representation of musical form for a 10 minutes performance from the Algorave 10th anniversary party. The radius shows an acoustical feature that is an indicator of event density, whereas the heatmap shows an indicator of perceptual brightness. Each slice represents a duration within the limits of our short-term memory (up to 8 seconds).

One of the key points of the study is that the analysis of the musical form is informed by music perception and cognition. The focus is on our short-term musical memory, which has an indicative time span of 0.5-8 seconds [99]. This is the main informative chunk concerning the visual representation of the musical form, as we compute a global descriptor for every short-term memory window, as shown in Figure 5.2. Each slice on the plot represents two acoustical features in the case of Figure 5.2, a rhythm-related feature corresponds to the length of the radius, and a pitch-related feature corresponds to the heatmap.

Research on musical structure in live coding is relatively unexplored, perhaps partly due to its improvisational nature. Few current studies are focusing on musical structure [180], [181]. Dal Rì and Masu explored linear and particle-based (density plots) as part of live coding practice study, and Magnusson

implements a circular representation of musical structure in his performance system, Threnoscope.

Sound visualization may be seen as complementary to code visualization, although how code renders processes to sound is not straight-forward. The visualization of code is important in live coding. That has been a point of attention since the early days of live coding, most notably in the work of Dave Griffiths and in the Al-jazzari, Betablocker and Scheme bricks systems. Today, more tools offer advanced features for visualizing code, most notably Strudel[6] [182] and Gibber[7] [183], but still, the focus seems to be on *instantaneous visualizations* of musical events of signal generators. Code visualization is seen as a facilitator of code comprehension when applied to secondary notation and as providing meaning enhancements when applied to primary aspects of the syntactic rules [184].

The circular plot in Figure 5.2 enables us to quickly recognize significant changes in the musical structure, such as a *climax* in the event density or the pitch height. Furthermore, larger musical constructs, such as *motifs*, may also be spotted. Such a visual representation describes the musical outcome and can make the performer aware of the evolution of their own performance. As such, it is an informative manner that may also be seen as descriptive notation. In live coding, the power of descriptive notation may be greatly appreciated, as has been demonstrated in several systems, like Flock [185, chap. 7], Threnoscope, Strudel, Gibber, and more.

Our work offers a new window into how we may connect cognitively-informed musical information in our improvisation sessions. Only vigorous experimentation can answer whether and how such technologies would be useful for the audience and performers. The performer tends to share everything with the audience, and it can be difficult to assess whether such transparency can benefit both parties. Accessibility is important but it may be that audiences do not want to follow everything that is happening on the performer's side.

5.3.2 Conceptual framework for designing agent-based systems

I introduced a conceptual framework in Paper VI [39], in the form of a non-linear and modular map to suggest to the community how agent-based designs can be applied to both the written code and the musical outcome (Figure 5.3). It is an observational study, where 8 systems are the focus. The study presents an *incomplete roadmap* for employing machine listening and machine learning technologies during a live coding session. I say "incomplete", because several more modules could possibly be added, as I mainly discuss a limited number of *affordances* and *temporal constraints*. Over the observed 8 use cases, I undertook a minor focus on code generation techniques and even less focus on *online* machine listening technologies. Whereas most of the systems put under examination do perform some computational analyses on the code or the results of the running processes, only two out of the 8 systems apply modifications to

[6] `https://strudel.tidalcycles.org`
[7] `https://gibber.cc`

the prescriptive part of the code, and only one more can produce novel code chunks for a virtual performance partner. Given the technological advancements in large language models (LLMs), it might be a matter of time before more and more systems will incorporate code generation techniques and program synthesis, but this is yet to be shown.

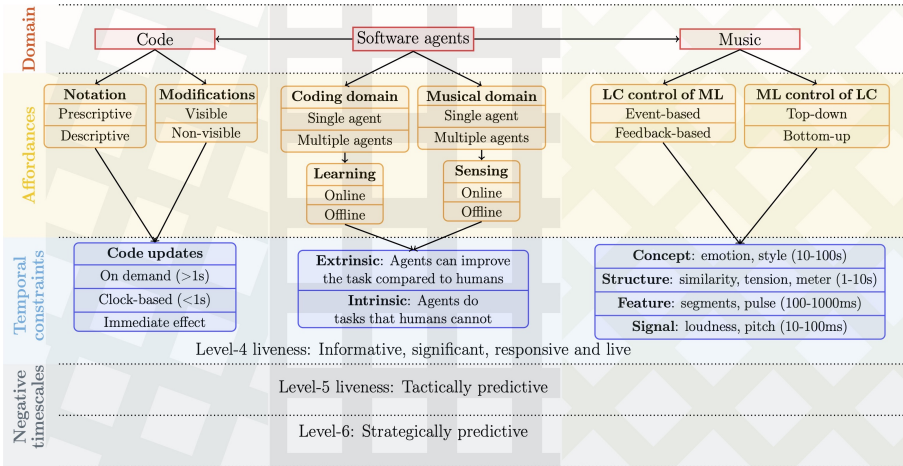


Figure 5.3: Conceptual framework for designing agent-based systems.

Also, a main interest of Paper VI is the technical understanding of liveness, as presented by Tanimoto's [43] hierarchy of liveness in programming environments. A comment on this technical dimension of liveness is that it is rather striking that advanced levels of liveness (i.e., level 5 and level 6 liveness) have been poorly explored in live coding. The reasons for this fact are unclear, but it may be that we find the comfort of *ebb and flow* in what Tanimoto calls level-4 liveness.

5.3.3 On liveness in bottom-up systems

As mentioned above, all performance systems in live coding are necessarily level-4 liveness systems. Still, to the best of my knowledge, there is no system to date that claims to be a level-5 system. Paper III [40] introduces a redesign of the stateLogic machine, which points towards a level-5 liveness system. In the git repository (`https://gitlab.com/diapoulis/lhc-knob`) can be found an interactive implementation of a system described in 2022 with level-5 liveness. The system implements *code completion* and can be considered a level-5 liveness system, although I did not implement any advanced predictive algorithms as suggested in the article. The system provided in the git repository uses nothing more than regular expressions, but still presents a case where the user is informed on the future behavior of the system.

An important note is that the system we proposed is not Turing-complete, rather it generates a formal language by employing regular expressions. In such a manner, the formal language is generated on-the-fly. Specifically, when

the system starts, there are no words generated. The vocabulary of the specific formal language presented in Paper III has a total of 7 words. Unless a new word – also known as token – is generated, it is not part of the language. How such minimalistic designs converse with liveness is yet to be seen, but my guess is that multi-agent systems that form networks of minimal agents is the way to go. Each agent could specialize in a specific level of abstraction and perhaps their collective effort would bring about emergent outcomes on language generation practices.

5.4 Implications

Here, I present the findings as "modalities" to address some of the important outcomes of this study, along with my vision of how we can move forward. The modalities refer to different ways in which we may apply creative applications. My vision builds upon agency, which is seen here associated with predictions. Several creative applications build on predictive modeling, and our expectations are formed by our sensorimotor predictions. Thus, my focus is on how such technologies can help us in such an endeavor and what points require careful consideration.

I see three main modalities of interest for creative technologies in musical live coding; *sound*, *visualization* and *text*. Sound goes without saying for musical practices, and its significance was demonstrated in the month-long practice sessions with the no-listening condition [38]. The importance of text also goes without saying in live coding, but my focus here is on the technical dimension of liveness. Visualization tools are discussed from two viewpoints: i) how to visualize sound and ii) how to employ visualization as conceptual tools. The first point has aesthetic implications whereas the second point is more akin to problems in information visualization.

5.4.1 Sound

In this section the discussion is centered around machine listening and human listening. Cognition-informed decisions are important for both, as auditory perception already validates machine listening. The focus is how to use sound as a creative technology.

Creativity support technologies that can be applied to sound are largely related to machine listening. In Paper VI [39], I present the affordances and temporal constraints of machine listening technologies (Figure 5.3). We can apply machine listening either in an event-based fashion or by using feedback. If the machine listening we do *controls* the live coding session, it can be done in two ways; either in a top-down manner or a bottom-up manner. By top-down, I mean we program using high-level programming abstractions, whereas in bottom-up, we use low-level programming abstractions. The temporal constraints in Figure 5.3 are self-explanatory, indicating the different temporal constraints where we can apply various machine listening tasks. For instance, pitch formation occurs within approximately 10 milliseconds, while evoking emotions when listening to a song can take several seconds or even minutes.

My study in practicing live coding for one month (Paper IV) shows that the absence of sound can also be used as a creative technique. Essentially when not listening we cannot experience how our coding modifications sound, which can be seen as an intentional *defamiliarization* with our auditory expectations. While this may be surprising at first glance. But when not listening to the musical outcome, we may be able to better focus on the text, and also try novel ideas without the anxiety which we may arise when we apply something really radical. When listening to sound we tend to avoid code modifications that are likely to produce an extremely loud or extremely unpleasant musical outcome. On the other hand, when there is no listening, certain tasks become impossible, like the perception of event density, the crafting of musical transitions, the perception of tempo, and more. To this end, sound visualizations are helpful, but such information about sounds levels and the spectrum are not necessarily perceptually-validated, depending on the task. If the task is to inform us whether there is audio output, then a simple sound level visualization may be informative. But when we want to get an understanding of more subtle qualities of the musical outcome, such as timbral characteristics and the perception of rhythm, it can become impossible when we only have available sound visualizations, such as of the spectrum. Therefore when there is no listening, higher-level visual representations such as latent spaces or cognition-informed visualizations can be useful.

5.4.2 Visualization

In live coding there are two main visualization strands: we want to visualize either the code or the sound outcome. The work presented in Paper V [44] demonstrates how to visualize the musical outcome using a circular representation. Time, in music, is usually represented either linearly or periodically. Modern tools, like Strudel, have introduced circular-based visualization (i.e., `spiral` command) as alternatives to linear representations such as the piano roll. The above mentioned circular representation is informed by cognitive processes, specifically our musical memory, and we noticed that Gestalt principles such as good continuation are validated.

Also, we presented two conceptual frameworks where we used visual representations to communicate the findings. In Paper I and II [35], [36] we employed a three-dimensional representation to demonstrate visually how systems look alike and differ regarding gestural control. I did not include these two articles in the Table 1.1, because they are mainly conceptual articles and do not have direct implications for this chapter (Study C). In Paper VI [39], we used a non-linear map, where the systems are mapped using a continuous trace-line connecting the different components. In both cases, visualizations are used in a manner that helps our understanding and thus can be seen as *conceptual visualizations*.

5.4.3 Text

Another modality that is necessary in the standard paradigm of live coding is that of text. This view is indeed restricted to particular facets of live coding, as visual programming languages like PD and MAX/MSP can only be partially included in the textual modality. Thus, notation is a more appropriate general term, but I do not use it here because it can also be seen as a visualization technology.

As discussed above, creative tools related to text, such as code previews, can disrupt pace and flow. In my personal experimentations with bottom-up live coding, I found that *code completion* is useful, as it eventually becomes obvious which regular expression will be excluded from the lexical analyzer – although I am not typing text on a keyboard, I generate text. Is it the case that code completion is only useful in bottom-up live coding systems? This is really an open question, and we will not have the answer unless we experiment more and more with advanced levels of liveness in live coding practice. With the current advances in LLMs it seems unavoidable that more and more systems will incorporate code generation and program synthesis. It remains to be seen how live coders will incorporate such tools in their practice and how this will change the current state of practicing live coding.

5.5 Publications in relation to Study C

In this study the related articles are shown in Table 1.1. Study C is a mixture of methodological and applied contributions. The most relevant articles (Paper IV and Paper VI) are both mainly methodological contributions with some applied content, as shown in Figure 1.1. The month-long practice study (Paper IV) contributes to the discussion on GUI helpers and sound visualizations [38]. The study on agent-based systems (Paper VI) contributes to our understanding of agent-based systems and liveness technologies [39]. Both articles provide practical code examples. In the case of Paper IV, I provide an extensive collection of live sessions as a git repository. In the case of Paper VI, I provide a code example for SC3 with the aim of giving novice users an understanding of how to implement simple software agents in their performance systems. Paper V [44] contributes a cognition-informed visualization for offline analysis of musical form, but we suggest that it may also be useful for online performance practices. Finally, Paper III [40] contributes to the discussion on liveness and delivers a git repository with a practical implementation of *code completion* for bottom-up systems, which possibly presents one of the first known cases of a level-5 liveness system implemented for musical practice.

Chapter 6

Study D: Reproducibility, transparency and risk

> "It's risky to just run code! No debugging or testing is available."
> Nick Collins, Alex McLean, Julian Rohrhuber and Adrian Ward [22, p. 322]

6.1 From the general to the specifics of reproducibility, transparency and risk

The interactions between reproducibility and transparency in the sciences can be seen as a risk factor in the maintenance of scientific integrity and scientific progress, and is often described as "the reproducibility crisis." In science and engineering, risk studies may range from risk management and risk assessment of software or hardware infrastructures to risk factors in medical research and social dimensions of job insecurity, to name a few. Thus, risk is a scientific research field and relatively few philosophers have been focusing on it [186]. I will be focusing on aspects of *risk* as a qualitative manner in relation to musical live coding.

In live coding reproducibility is a notion that is applicable only to notation. Code is seen as notation and can be fully archived using time tags, so that a live set can be reproduced. That makes live coding different from most improvisational practices, as it is hard to claim that an improvisational practice is reproducible. On the other hand, transparency is deeply embedded in the live coding community, primarily as an feature of the open source and free software culture, but also stemming from the improvisational nature of musical practice itself, which is welcoming of "imperfect" performances. The live coding ethos is depicted in the TOPLAP motto "show us your screens", but also in the first book published on live coding by practitioners and active authors in the field, which uses a copyleft license [25, p. xv]. Risk in live coding extends beyond the possibilities in an instrumental improvisation practice [22], and is also associated both with technical and musical aspects of the work [28],

which can range from performance setups and musical aesthetics to language, notation [187], system design and more. The general attitude of the live coding community towards risk is that it is an indispensable ingredient of live coding performance. Indicatively, the first known live coding performance, by Ron Kuivila at STEIM in 1985, resulted in a system crash [188], [189].

6.2 Reproducibility, transparency and risk in live coding

6.2.1 Reproducibility in live coding

Live coding practice is largely indifferent to reproducibility. Whereas the notation of a performance can be recorded and replayed, this is not a common practice for live coders. It is rather the case that the code is ephemeral. After a live set, the script is not necessarily stored, and this is one of the main intricacies of live coding that makes the practice attractive to many practitioners. Thus, reproducibility addresses only archiving and analysis procedures, which are largely not common practices in this work; this may be an attribute shared among many improvisation art practices.

In the first International Conference on Live Coding (ICLC), *live writing* [116] was introduced as a means to reproduce, character-by-character, the live typing of the code. This approach may radicalize our perception and understanding of live coding practice, as it raises questions on telematic and collaborative sessions, where network delays can be hard to trace, in order to accurately reproduce the exact experience. This raises philosophical questions about the multiplicity of future outcomes of a telematic performance, as every performer of a telematic ensemble is experiencing their own temporal reality. Of course, we have been aware of this fact, but live writing demonstrates in practice how these multiple realities unfold to create a unique performance.

6.2.2 Transparency in live coding

An immediate criticism of live coding transparency procedures comes from their sometimes obscure practices and is linked to what may be called technical transparency. A complementary notion is that of *selective transparency* as we "select" what to show to the audience [190]. Projecting the screen to the audience does not secure transparency for the audience, regardless of whether they are expert or untrained live coders. Obscure code can induce in the audience a sense of uncontrolled and chaotic practices and lead them to wonder about the performer's intentions [26]. This is truly an issue in live coding, as many practitioners, including myself, mostly perform using pre-written code. I admit that in my practice, scripts overloaded with code can be challenging to read and navigate. Live coding is known to have a "terrible closeness to mapping" [188], a term to denote how the output is tightly linked to notation [191], and in my practice, I often use lengthy one-liners[1]. That makes forming a mental model

[1]Sometimes also called sc-tweets, sc-twitts, or sc-toots, as if they were standalone tunes.

about the running program even harder. It can be difficult to navigate through the code, understanding what effect different code chunks are having on the musical outcome.

While these can certainly be seen as bad practices, it comes naturally for me and other practitioners to write lengthy one-liners. Maybe this is an outcome of practicing how to write sc-tweets, which are musical compositions presented as one-liners in SC3 code and published in Twitter[2]. Writing an sc-tweet is essentially an iterative design practice, and is fully transparent in the sense of selective transparency (Figure 6.1). Before publishing an sc-tweet, there may be several dozens, or even hundreds of iterations. Indeed Fredrik Olofsson estimates that an average of four hours is spent on each sc-tweet [95], and provides in his cookbook some indicative rules each sc-tweet should follow. Even the tuning of the synthesis parameters is an essential aspect, as to make a compact code small enough to fit the old Twitter upper text limit of 140 characters is a challenging endeavor. So, while one-liners can be a bad design decision in terms of code comprehension and readability, to people writing sc-tweets this is seen as a character-saving (text-economic) practice. Of course, when the one-liners are lengthy and cover several lines of code on the text editor (more than 250-300 characters), navigating through them to apply modifications can be challenging and sometimes impractical. I have seen a performance by Fredrik Olofsson[3], where he uses three lengthy tweets to begin the live set. I found this a clever approach to starting a live session, because if you simply constrain yourself to use as little as three different tiny programs, you can study the code and manipulate it effectively during a performance. Starting a performance with snippets of pre-written code is also highly appreciated by the community, as in a blank slate coding session, the progress is rather slow, and the musical outcome may not be musically pleasant. So, taking care of what the audience will listen to is also highly appreciated by the community, although this comes with the drawback of poorly exploring complex algorithms during performance time [104].

6.2.3 Risk in live coding

A descriptive definition of risk is: "situations in which it is possible but not certain that some undesirable event will occur" [186, p. 2]. The different qualities of risk between traditional and live coding music performances are numerous, from the generativity of the musical outcome in a live coding session to the fact that a musical instrument will not melt down during a performance[4]. Live coding has been described as real-time instrument making [53], and a

[2]There was an online release of SC tweets in 2009 `https://www.newscientist.com/article/dn18173-best-of-twitter-tunes-album-released/`, `http://mcld.co.uk/blog/2009/sc140-squeezing-entire-pieces-of-music-into-tweet-sized-snippets.html`.

[3]Algorave 10th Birthday March 2022 - redFrik - 2022-03-19 22:10 `https://youtu.be/lqi-Vqr0qk4`

[4]Ron Kuivila's, first known live coding performance, has been described by Curtis Road as "I saw Ron Kuivila's Forth software crash and burn onstage in Amsterdam in 1985, but not before making some quite interesting music. The performance consisted of typing" [189, p. 249].

Figure 6.1: SC tweet exhibition by Charles Celeste Hutchins.

programming error may produce persistent effects on the outcome [28]. If this happens during a performance, it is not uncommon for the coder to restart the system. Less significant programming errors may result in transient disruptions or glitches, which the coder may be able to resolve on-the-fly.

Knotts [28] has classified errors as either *transient* or *persistent*, to indicate the difference between a system crash and programming errors that can be handled without stopping the performance. Other risky behaviors may relate to the musical outcome and have no relationship to programming errors. For instance, it is possible in musical interfaces that use generative algorithms to explore a sound synthesis space, which suddenly tunes its parameters to silence [125], [127]. This problem may extend beyond the scope of live coding. Thus, the expression of risk in generative music systems is not necessarily related to programming glitches or errors but may extend beyond mere technicalities and can be related to the musical outcome.

Abstract architectures have been described as designs that can increase risk, as the interface cannot take into account practical concerns about the intended application [192]. Abstraction is an inherent quality of all live coding performance systems, and it is not uncommon that the desired requirements are not exactly met during a performance. This is known as *abstraction management* in the cognitive dimensions of notation [187].

Cognitive and motoric risks are also involved in live coding performance. It is well known to live coders that performance anxiety may result in "shaky

hands" [193]. McLean argues that stress during a performance is not an unwelcome feature, but this has to be on moderate levels so that it does not induce destructive stress [194]. Several more factors may be present as risk factors. Here, I am considering the sensorimotor constraints imposed on the coder due to their typing activities. Typing on a keyboard is an activity based on serial skilled actions. It is an open-loop motor control process returning no feedback from the performed actions [37]. This state of affairs may induce significant typing errors during a performance, inhibiting pace and flow.

The audience appreciates the effort put into a performance, especially when significant risks are taken, according to Roberts and Wakefield [104]. Collaborative performance is seen as an activity to limit and control risk factors during a performance. For instance, when a duet is performing, and one out of two performers lapses into silence, the other performer most likely will still be generating musical sounds. Similarly, when there are more than two coders, the possibility of a total crash of the system is not likely to happen. That can be seen in analogy to a property of network architectures, wherein, when a node goes offline, the remaining nodes may still be fully operational depending on the network connectivity.

Overall the aesthetics of failure are much appreciated by the live coding community. The idea is that live coding is a *non-productive* programming activity, rather a fun activity where coding is seen as a leisure activity, free from technical innovation, robustness or stability [28].

6.3 Contributions

6.3.1 Reproducible musical analysis: Algorave case study

In Diapoulis and Carlé [44] we conducted a reproducible study on the Algorave 10th anniversary party. The algorave culture is only a subset of live coding practices as shown in Figure 6.2 (original resource from the TOPLAP forum[5]), that focuses on algorithmically produced music for dance parties. The study's focus is music information retrieval (MIR), where we extracted various acoustical features from 133 live sets to examine the musical structure of the performances (Figure 6.3). The main output of the study is a circular representation of the musical form which may be used for interactive experimentation and exploration of the evolution of the sound during a performance. It is the first time that a MIR study has been conducted on such a scale in live coding research, and one of the few reproducible frameworks within the community.

The reproducible framework was built using Docker containerization technology. The docker image[6] is made as a modular build, so that it can be built in independent stages, and not as a monolithic build. This has several advantages as the overall process includes four main phases: i) retrieve the audio-visual

[5]A Venn diagram that show the relationship of algorave parties to live coding, figure adopted from *heavy-lifting* post `https://forum.toplap.org/t/the-cultural-differences-between-live-coding-and-algorave/673/6`

[6]Pre-built image on DockerHub `https://hub.docker.com/r/algorave10/iclc2023` (accessed 2023-08-26).

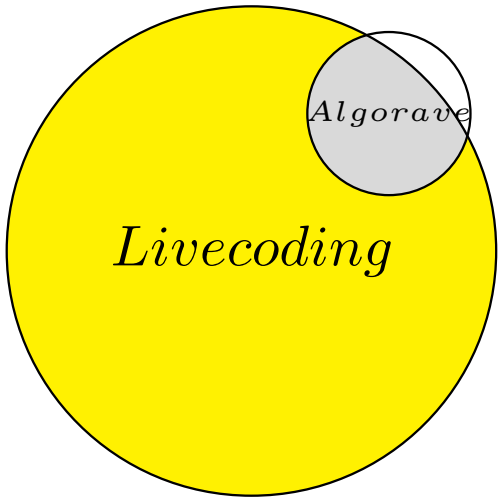


Figure 6.2: Algorave in relation to live coding. Figure adopted from the TOPLAP forum[5].

dataset from a git repository for large files, ii) transform the video data to audio, iii) extract the acoustical features for each performance, and iv) perform the descriptive statistical analysis. All four of these phases can be individually built even if any programming errors occur during the process.

The computational analysis provided by the Docker image was conducted in a Jupyter notebook, a popular approach to literate programming running on the browser via a localhost server connection. The analysis offered is focused on descriptive statistics. Our main concern was to deliver an easy and accessible approach to reproducing the information retrieval and data analysis, along with conserving some historical and statistical information about the community. Each performance had self-reported geographical location and programming language or system used, which are both indicators of cultural diversity, either musically related or socially related.

Another important goal of the study was to build bridges between academic communities and cross disciplinary boundaries in music research. As such, the study addresses readers from music cognition and music information retrieval to the live coding community and other underrepresented musical communities.

As the study only focused on the audio dataset, further work can be done on the visual data set, mostly screen recordings of the programming practices used. While it can be technically challenging to interrelate the audio data with their corresponding visual counterpart, this is not impossible. The difficulties arise due to the variety of programming languages used; most of the performances were conducted using TidalCycles. As a result, the dataset is unbalanced in terms of programming languages, which is certainly not an insurmountable problem but rather a challenging characteristic of the specific dataset. On the other hand, *selective transparency* [195], a term to denote what the performer is showing to the audience, can be devastating when aiming to

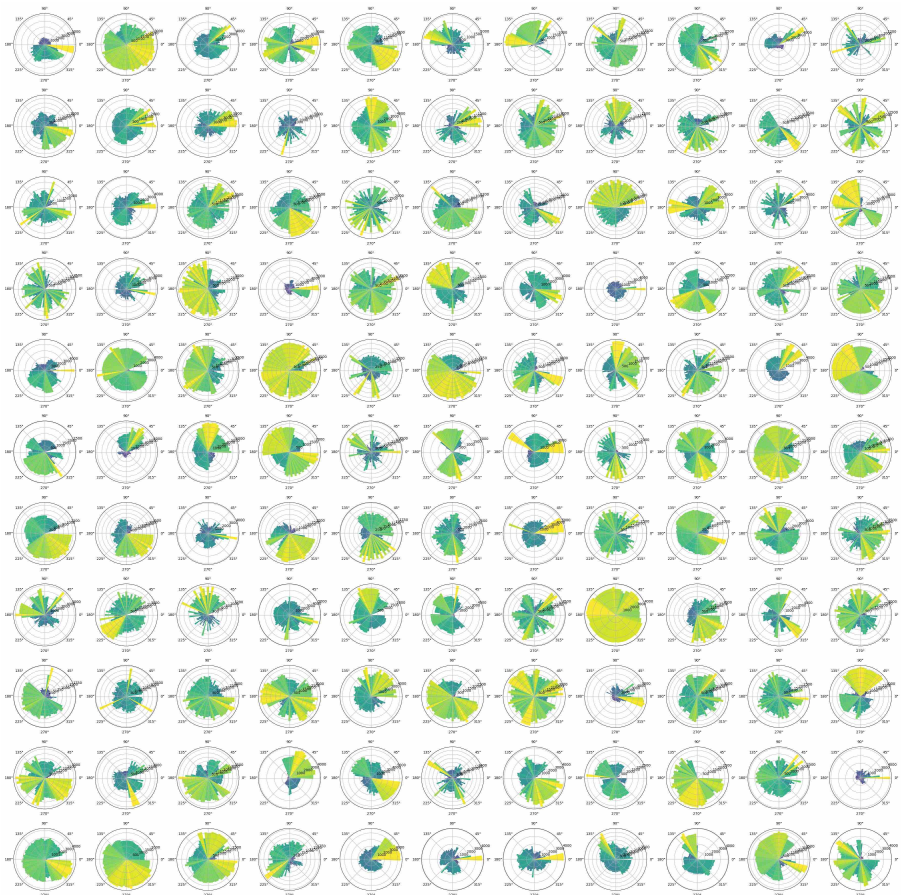


Figure 6.3: Circular representation of musical form for all 10 minutes performances, excluding outliers, from the Algorave 10th anniversary party.

find correspondences between audio and visual data. Here, there is no easy solution, and it is also the case that the 133 performances make a relatively small dataset for applying large scale learning algorithms. To address this issue, I would claim that not all the available live coding recordings to date are enough, and given the diversity of programming languages and musical practices in play, it is an open question whether this is even possible.

6.3.2 Transparency

During the month-long daily practice sessions [38], I was writing daily reflective diaries. The diaries are published on the git repository (`https://gitlab.com/diapoulis/livecodeme/`) along with the `History` files, a time-tagged file for SuperCollider. The history files can be used to reproduce the sessions, and the diary aims to make the study process transparent to the community. While

publishing reflective diaries may not be a common practice, there are certainly many live coders that use them as cognitive prompts (e.g., Sam Aaron [196]). This can be extremely valuable to the community, as the reader can relate to their practice and form certain expectations from such an endeavor.

The study contributes a unique methodology and rich data set. In total, 101 history files were produced, 62 from the experiment proper, 36 from the pilot study, and 3 more from two days before the beginning of the month long practice. Furthermore, there are 31 diary entries from the experiment proper and 12 entries from the pilot study [7].

The reproducible MIR study [44] contributes a structured data set with audio and video files on a git repository for large files[8]. Also, the docker files are stored on GitHub[9] and the prebuild image[10] is provided on DockerHub. This collection of code and data repositories presents a rich data set that demonstrates the unique diversity of the live coding community. The descriptive nature of the computational analysis aims to highlight the broad diversity of the different practices by incorporating a creative approach to visualize the musical form of a live coding performance.

6.3.3 Risk

Reproducibility aims to reduce risk factors, whereas transparency may increase them. For instance, by providing an open and reproducible framework, we aimed to help the community reproduce the analysis results. We focused on eliminating any cross-platform dependencies and delivered a docker image. Thus, we focused on reducing the risk of a non-reproducible outcome to the best of our capacity. In a sense, the abstractions we use in live coding, like a function or a class, aim to be reproducible. Otherwise, we risk failing if we write high-level programming constructs from scratch during a performance.

At the same time, offering an open data set can be seen as a risk factor for the community. How will this data set be used in the future? For what purpose? Can it be used to create stereotypes? These are open questions that no one can definitively answer, but only speculate about the multiple possible outcomes. This is the cost of transparency. If everything is open, someone can take advantage of the situation to their own benefit, either by stealing the artwork or other monetization opportunities.

Risk in live coding also has a different facet. I have identified that bottom-up systems are not subject to the risk of failure. All bottom-up systems, except iMac Music by Jonathan Reus, were designed in such a way that they cannot fail during a performance. The question arises, how can we make these systems more prone to failure? As already discussed, failure is highly appreciated in live coding. Here, I question whether pre-reflective processes can induce risk in bottom-up systems. Of course in the case of bottom-up systems that are not risky enough this would not make them crash on stage, but it still is an

[7] GitLab repository: `https://gitlab.com/diapoulis/livecodeme`
[8] `https://github.com/gewhere/algorave10-large-files`
[9] `https://github.com/gewhere/iclc2023`
[10] `https://hub.docker.com/r/algorave10/iclc2023`

aspect that increases the risk of successful code evaluations and performance in general.

6.4 Implications

6.4.1 Visual representation of sound

Music offers numerous opportunities to explore real-time scenarios and make a bridge between audition and vision. Music is a highly interdisciplinary endeavor that touches on musical notation, auditory display, crossmodal interactions and information visualization. For non-real-time applications, there are numerous ways to visualize data, both in plots and transformations. For instance, we can apply linear transformations to visualize synthetic dimensions, also known as latent spaces, which can be more informative than the original dataset. When doing so, some high-level interpretation of the synthetic space is typically conducted, which assigns meaning to our visualizations. For the circular plot of Figure 5.2 we may apply such linear transformation and visualize synthetic dimensions instead. For instance, we may cluster the radius representing rhythmical information and the heatmap representing pitch-related information.

Cognitively informed visualizations in music can certainly be valuable, especially during performance. Our work on the musical structure of the Algorave performances shows that when certain principles of musical memory are taken into account, Gestalt principles emerge, allowing us to perceive higher-level characteristics of musical form, such as motifs and themes. This cognitive strategy – using low-level characteristics to construct or predict higher-level characteristics – is well known to cognitive scientists and psychologists, and used extensively in experimental designs.

There is certainly a division between instantaneous and persistent visualizations in music. For instance, the spectrum is an instantaneous representation of the spectral content, whereas the spectrogram is a more temporary, though persistent visualization of the spectral content. A visualization that spans the whole duration of an improvisation session is not a common approach, and it also is impossible to make use of such representation when the duration is not predetermined. However, certain conventions are usually set in place within musical improvisation and it is not uncommon during a music performance to use a timer to inform you of the running duration. In these scenarios, I would claim that visual representations over larger temporal chunks can be informative of the musical aesthetics of the performance. In this manner, we may want to "reproduce" the musical aesthetics of performance, and visualization can be used as an informative tool on the performer's side.

In the case of live coding, visualization of the musical structure is not used, other than in the case of the Threnoscope, which demonstrates a slowly evolving persistent visualization as a descriptive notation. From my practice, I can assert that when I am focused on writing the code, I tend to ignore any other information that is on my screen. For instance, when live coding, I always have the spectrum and the sound level meters on the left side of my screen, but while in deep focus, or using strong mental effort I completely ignore what

is happening in the visualizations. Several practitioners and systems do embed the visualization as a background on the text editor, most notably Strudel, which I find a useful way to tackle this issue.

6.4.2 Live coding is risky

6.4.2.1 Risk of reproducibility and transparency

The interactions of transparency and reproducibility should be treated with care as a risk arises when tools become the artifact and not the vehicle of our creative practices [56]. Thus, any work that aims for universal outcomes should be aware of the dangers put into this pursuit. When a practice is detached from its consequences to its immediate environment, then we endanger losing our agency. Simply put, if a live coder replays the code recordings of my live coding sessions during the month-long practice, this will not improve his or her skill in live coding. Rather, they have to carry out a month-long practice to benefit from such an endeavor. This may be a simplistic and obvious example, but it denotes the risks associated with aspects of agency. A similar scenario applies for the reproducible study and the open dataset.

6.4.2.2 Risk and gestural interactions

The implications of risk in our gestural interactions with bottom-up live coding are central to this thesis. I identified that almost all systems are indifferent to risk. Simply put, bottom-up systems do not crash during performance. I find, here, that there is an opportunity to reintroduce risk in our gestural interactions, which means that the bodily gestures we do during a performance can fail. If we keep safeguarding our systems, we may end up in a situation where we have created a "domesticated" version of live coding. This would feel more like what I would call using a "generative bleep-box" than actually to engage in live coding.

6.5 Publications in relation to Study D

In this study, the related articles are shown in Table 1.1. Study D is a mixture of mostly methodological and applied contributions, with some influence from theoretical studies. Paper V [44] contributes the reproducible and open-source framework for analysis of the musical form of live coding sessions. The framework builds upon an open dataset of 133 live coding performances, and we delivered a technical solution for better accessibility over this dataset. Paper IV [38] contributes 101 live coding sessions that can be replayed in SuperCollider, along with daily reflective diaries. As such, both Paper IV and Paper V contribute to the questions of scientific and artistic transparency. Paper III [40] somehow silently contributes to this study as it also delivers an open source code repository, but most importantly, aids Paper VII [37] in its discussion of gestural interactions in bottom-up systems, thereby helping our understanding of how risk is an important factor for bottom-up systems.

Chapter 7

Discussion

The discussion is divided in two main parts: 1) a review of the contributions, and 2) implications and reflections. It is essentially a synthesis of the studies along with the presentation of the theoretical background, and I aim here to highlight the most important parts of this thesis. Preliminary discussion on the future work is presented in the present chapter, but there is section 8.3 of the Conclusions section which discusses the future work in more detail.

7.1 Review of contributions

7.1.1 Making bridges: Theory making for live coding

From the beginning, I have intended to bring to the live coding literature an approach to understanding how a music psychologist might react to this arguably unconventional musical practice. I contributed a sensorimotor theory of musical live coding that reflects on practice and can be used as a first step toward an embodied understanding to be used for studies in music performance. The main contribution here is the observation that typing is a series of skilled actions performed on a keyboard. This obvious and simple observation that came out as a combination of the literature review in studies of music performance [86], [123], reflections on practice, and observations of other practitioners, opens new horizons on the construction of a theoretical framework for live coding.

7.1.1.1 Music perception and cognition

Expressive interaction is a term that refers to the aesthetics that are induced as an outcome of our sensorimotor control. It addresses planning, coordination and anticipation [197], but also addresses agency and emotional reward mechanisms [31], [198]. I explain here how expressive interactions relate to live coding. Throughout the thesis, I use the term *pre-reflective processes* to describe a phenomenological approach to study fast processes, those that correspond to the lowest-level of motor control. The motor control system of humans can be

seen as a three-fold blend of low-level, mid-level and high-level control [197]. Sensorimotor synchronization (SMS) is carried on the low-level, coordination on the mid-level, and planning on the high-level. This is a hierarchical model that covers different time spans. The next unexplored step to better understand cognition in live coding is to study aesthetic enjoyment and emotional responses, per Palmer [86] who discusses how expressive performance is related to musical structure, bodily motion and emotions. Aesthetic enjoyment up to the present has been discussed in terms of audience appreciation [199] and the interplay between audition and vision, and a few studies focusing on the aesthetics and evaluation of computational creativity [200], [201]. The contributions regarding musical structure are limited to a few studies, each focusing on a different methodology and presenting a broad understanding of the term [180], [181]. To this end, our study differs as it presents a cognitively-informed visualization of the process [44].

A conceptual framework was presented here that discusses how live coding is related to various musical activities, that is, of music-making, music listening, and the formation of musical imagery [35], [36]. This discussion offers a new perspective on musical practice as it discusses the role of music listening and musical imagery during a performance. Music listening is crucial and seen as the "glue" that allows us to perceptually bind the written code with the generated sounds. Listening is linked to anticipation, and imagery to planning. In that sense, listening can help our predictive mental models – thus, empower our agency – whereas imagery is linked to higher level mental operations. Musical imagery is crucial when we write and merge in existing novel code chunks during performance, and is linked to musical notation via notational audiation [37].

During music-making, multiple levels of cognition come into play and enable us to reason and develop an understanding of the relation between the code and the sound. A feedback loop between listening, imagining, and making music ensures that we link perceptual continuities to cognition and reasoning. Precisely how this is done is fairly unexplored, but I have discussed the importance of notation in live coding and how it may be linked to our sensorimotor network.

This thesis offers extensive discussions of our various cognitive and perceptual limitations, and the reader can find here best practices for the design of performance systems. For instance, extensive discussion on sensorimotor control is provided in Paper VII [37], along with a general theoretical background on musical imagery, that expands in two articles (Paper I, PaperII) [35], [36].

The thesis also examines music listening and musical imagery in live coding, one in relation to another. This opens new pathways on how to study live coding as an activity of everyday life [202], such as in *live coding outside* [203]. Research in this direction can provide a perspective on the aesthetics of the everyday in live coding.

7.1.1.2 Gestural control and the meaning of bodily gestures

I have discussed the idea that typing during live coding is a kind of musical gesture, which are actions that assign meaning to our practices [35]–[37]. Baalman exemplified this practice in her performance Code LiveCode Live, which resolves long-standing issues on whether typing on a keyboard is actually a musical gesture. Furthermore, her specific performance is the first known case of the appearance of sound-producing gestures in live coding. Later contributions include CodeKlavier CKalcuλator and essentially all other systems that involve a traditional musical instrument.

Live coding is a practice that mainly facilitates secondary aspects of musical gestures with communicative and expressive content. Given that the study of musical gestures is an established research field across various music-related disciplines (e.g., music psychology, NIME), understanding bodily gestures in live coding and their limitations can result in new interdisciplinary collaborations. This is not applicable only to live coding per se but applies also to the broader electronic music field, particularly that of generative music. Music psychologists have formerly tended to focus only on traditional expressions of musicianship.

7.1.2 Bottom-up live coding

A bottom-up system was presented in Paper III [40], implementing a redesign based on a hardware and software prototype [41], [42]. From several of the observation studies conducted as part of the thesis (Paper I, Paper II, Paper VII) [35]–[37], it becomes apparent that all bottom-up systems share distinct characteristics related to either the crafting of hardware components or of formal languages. A distinction is drawn between bottom-up and low-level computing in live coding to clarify that the term is used differently across the literature, a fact in agreement with Roberts and Wakefield [104]. I have also used the term *low-level* to refer to low-level programming languages [40], [42], and in Roberts and Wakefield [104] the term is mainly used to indicate low-level computations related to DSP processes by various practitioners. Although, at the same time, the term bottom-up is also sometimes used to refer to progressively building abstraction levels. Thus, I suggest that due to the ambiguity of the term as it has been used in live coding, it may be useful to clarify the exact context of a use case. Whereas the bottom-up term stands for systems that afford programmable actions in the context of formal languages, low-level computing does not secure this integration but may simply point out digital signal processing (DSP) algorithms and other sound synthesis techniques used during a performance.

On the other hand, a similar argument can be made for the use of the bottom-up term, as one can apply sound synthesis processes in a bottom-up manner: that is, progressively building the unit generators to specify sound synthesis structures on-the-fly. Essentially, one can claim that all blank-slate sessions are a kind of bottom-up live coding. I would still claim that, in this case, the unit generators used for sound synthesis do not necessarily have a formal structure dictating their use. By formal structure, I mean no grammatical rules relate one unit generator to another. Such a bottom-up structure would most

likely not rely on a formal grammar, or be Turing complete, but it is rather a relatively unstructured routing and entanglement of different things. Thus, I would suggest that *bottom-up live coding* presupposes the existence of low-level programming constructs that are built and verified during a performance. I would envision that this low-level nature of bottom-up systems may be useful for approaching and exploring higher-levels of cognitive organization, such as coordination and planning.

The work presented here is centered around the notion of bottom-up live coding. It has been identified that such systems are still relatively rare. Although it is well-acknowledged within the community that easy classifications in live coding can be more of a problem than a solution [48], I have discussed the significance of bottom-up systems from both a theoretical and a practical perspective. In short, the importance lies in both the system's workings and the human coder's cognition. On the system-side the program becomes more modular, where each module is verified and can be ready to use, whereas, on the human-side, the users tend to use such systems in a swifter manner that is akin to traditional musicianship.

7.1.2.1 Predicting programming behaviours: A technical view on liveness

The redesign of the stateLogic machine (Paper III) presents one of the few cases in which continuous control is used in bottom-up systems. Earlier works include Approximate Programming and CodeKlavier CKalcuλator. Although the importance of both aforementioned systems is unquestionable, the redesign we presented goes a step further to address whether a tactically predictive system can be implemented from the bottom-up. In this thesis, I present the git repository (`https://gitlab.com/diapoulis/lhc-knob`) with a simple implementation of a tactically predictive system that performs code completion based on regular expressions. Although the presented implementation does not offer a predictive model between continuous gestural control data and code completion, it realizes how a bottom-up level-5 liveness system can be implemented using regular expressions.

Tanimoto considers the hierarchy of liveness as part of a larger category of feedback mechanisms in software engineering [147]. Nash and Blackwell extend this analysis of liveness, discussing the "quality of the feedback in different domains" [101]. The system we have presented in Paper III provides a fruitful testbed for experimentation and study of the continuous control of gestures with advanced levels of liveness (in this case, level-5 liveness). For instance, direct manipulation has been used in live programming [204], which applies code changes to an HTML/DOM file using the mouse. In our case, the system redesign implements direct manipulation, using a knob, to generate a formal language.

From a *psychology of programming* perspective, the redesign shifts the focus from aspects of *visibility* and *meaning* to facilitate *fluidity of actions* [119]. In live coding we often forget about the fluidity of actions, probably because we think of the keyboard as the "standard" input interface, although several

live coders put a lot of work into developing different types of keyboards (i.e., alternatives to the QWERTY keyboard). In any case, I think that predicting programming behaviors in interfaces that afford continuous gestural control would bring forward novel interactivities, which may be more rewarding and enjoyable during a performance.

7.1.2.2 The significance of pre-reflective processes

A general tendency for shifter gestural interaction is identified in bottom-up systems. Few of them demonstrate apparent activations of pre-reflective processes, as identified by the fast gestural interactions the user uses with these systems. Typically, these are bodily gestures rendered within a few milliseconds, and in one case (CodeKlavier), the coder is playing a traditional musical instrument, the piano in this case. A thorough study [37] of psychological and perceptual aspects of musical live coding examines various performance practices and systems, and discusses gestural interactions with these systems. The accompanying theoretical analysis reflects on practice, and provides cognitively informed decisions that the coder may take into account when designing performance systems with an embodied orientation.

7.1.3 Recognition and retrieval processes in interfaces and gestural control

Live coding demands retrieval processes from the performer, meaning that the coder has to recall from memory a wide variety of different things, like the programming commands and more. This is the case in both interface design and in programming notation. In interfaces, the live coder has to recall long sequences of individual characters that are required to type a single command, especially so in the top-down approach, which uses a keyboard and a typical programming language such as SuperCollider, Python or Lisp. Anecdotal evidence by Marije Baalman on typing automaticity suggests that a level of pre-learned motor patterns can serve to free cognitive resources; of course, this is likely the case only for short sequence a few characters long.

Contrary to the top-down approach, bottom-up systems mostly rely on recognition processes. This is also the case for the redesign of the stateLogic machine, but also systems like Betablocker, Al-jazzari, the TOPLAP app and iMac Music do share the same characteristics. In contrast, CodeKlavier CKalcuλator demands retrieval processes from the performer, a result perhaps of the complexity of the interface in HCI terms, as a gestural vocabulary has to be retrieved during a performance so that the system can recognize musical patterns (Figure 4.2, see Noriega & Veinberg). As such, CodeKlavier CKalcuλator may be more similar to the Gewording system, where the performer recalls the gestural vocabulary in order to write the program.

Even in the case of Orca, which has single-letter commands, the coder has to remember a list of 26 different commands. Knowing by heart 26 commands may be something that can be easily learned, but during the learning process it also requires that the coder make a conscious effort to recall the given

commands, as there is no descriptive association between the single letters and their corresponding semantics. To this end, Orca opens questions on the importance of familiarity and expertise with the user interface, as we can assume that extensive training may automate a variety of tasks.

7.1.3.1 Open-loop motor program and spatiotemporality

I mentioned above typing on a keyboard is an activity that relies on an *open-loop* motor program, which means that there is no *closed-loop* sensory feedback during movement execution. Simply put, there is no way the coder experiences any tactile or other sensory feedback during the typing of a single character. This means of control shares similarities with other electronic instruments, like the theremin and other air-instruments, although there are qualitative differences between continuous gestural control and *discrete-target control*[1]. The theremin presents an important case of electronic instrument, as it is the first known electronic instrument ever made. Modern day air-instruments vary to a great extent, and Jensenius [53, p. 242] remarks that air-instruments are based on "noncausal interactions."

The open-loop motor program presents several challenges and consequences, such as making typing prone to errors, as there is no feedback from the keyboard that the coder can take into account. When a key is pressed, the action is not retractable, and the only way to make any corrections is simply to delete and type again. This is the reason that air-instruments are so challenging to master: most notably the theremin, as during performance with a theremin the gestures are directly sound-producing, and every small bodily movement result in an immediate audification of the electromagnetic field. In contrast, in live coding – with the exception of Code LiveCode Live by Baalman – we do not make sound-producing gestures. Furthermore, when using air-instruments the performer has to deal with micro-, meso- and macro-spatiotemporalities [53, p. 233]. When typing on a keyboard, we are not largely subject to spatial dependencies.

7.1.3.2 Musical memory in live coding

Previous studies on the perception and cognition of live coding performance [141], [142], [150] essentially focus on long-term memory processes (longer than 15 seconds). Whereas this is precisely the case for most canonical live coding systems – those that use the keyboard and a textual language – this description falls short of characterizing systems that incorporate short-term memory processes (STM). The instances of STM activations are numerous, whether in ixi-lang, or in short edits to single-character commands, or in direct musical instrument playing or in using other interfaces that afford direct manipulation [37].

Bottom-up systems do offer many opportunities to integrate STM processes into live coding. As has been mentioned above, all such systems do afford a

[1] A term I use here to refer to the qualitative differences that arise when controls have a specific target, as in any teleological inquiry.

tightly-knitted temporal relationship between the coder and the system. In a few cases, this tight integration can even activate pre-reflective processes, thus making good use of the first level of our musical memory, the echoic memory that is activated within less than 0.5 seconds[2]. (Its general function is to inform us about our acoustic surroundings.)

7.1.4 Visualizations

We have discussed code visualization in live coding from different angles [61], [184], [205]. There are various techniques for representing various aspects of the music, such as particular parameter values, patterns, continuous data, the semantics of code, and musical form [44], [137], [180] among others. This thesis presents two uses for visualizations: i) to visualize sound, ii) to visualize a system. As for sound, we contributed a visual representation of musical form, and to visualize the system, we presented tools for conceptual visualization of creative applications.

7.1.4.1 Musical form

In Paper V [44], we constructed a cognitively-informed circular representation of the musical structure. This representation is based on short-term memory and is indirectly validated by the Gestalt principles of good continuation, similarity, and proximity which emerge as the outcome of the cognitively-informed design (Figure 5.2). We suggest that this visual representation may also be useful for real-time applications, as it can offer visual cues to inform the coder on the dynamic state of certain acoustical features. Furthermore, with such visualizations, climaxes and other higher-level characteristics of musical form can be easily spotted.

7.1.4.2 Creativity tools

A GUI helper showing a matrix of unit generators (UGens) was developed in conjunction with my month-long practice sessions [38]. This rather simple visualization aims to help the live coder when unclear about how to continue a live session. The aim is to transform the helper into a software agent that performs actions and enhances the written code with novel code chunks.

The conceptual frameworks in Paper I, Paper II and Paper VI, have the intention of facilitating the readability of the articles, but they also come out of a process of attempting to sketch out different concepts. Sketching is a creative technique used widely in HCI and can facilitate our imagery through the creation of visual representations [206]. An analytical framework of gestural interactions [35] introduces a three-dimensional representation of the systems' space. This cartesian plot (Figure 4.2) may be misleading, and at first glance can be interpreted as a framework for quantification and measurements. Instead, our aim was to demonstrate in a visual manner how various systems differ and

[2]For expert musicians we can assume that these durations are further extended through extensive musical practice, so we can claim that echoic memory may reach to a few seconds.

relate to one another. Higher-level diagrams are presented in both studies. The conceptual framework in Paper I is coupled within a higher-level explanation of musical live coding (see Figure 1), and the diagrammatic representations in Paper VI (see Figure 1-9) visually explain live coding as a musical activity and as a cybernetic system between a human and a machine.

The agent-based framework [39] presents a non-linear map of the system's affordances and the system's temporal constraints with the aim of helping practitioners to design systems that involve machine listening and AI technologies. I use a modular design for the conceptual framework and I map 8 systems, by visualizing the trajectories between interconnected modules. The framework may resemble a "roadmap" to various components that agent-based systems incorporate and its modularity invites further additions. Ultimately, the idea here is that the coder can learn to employ the map and make ontological relations between different domains [33, p. 136].

7.1.5 Structured datasets

A central practice of this thesis has been the conducting of online observations. A plethora of accumulated material online has been increasing dramatically during the last decade, and has opened rich opportunities for conducting ethnographic, anthropological, and music information retrieval studies. This thesis examined a wide variety of systems and practices. I spent numerous hours searching online documentation of original performances from individuals or grassroots organizations.

This longitudinal endeavor resulted in several structured datasets, which I present as "Observational datasets" in the next section. The reproducible MIR study contributes an open git repository of video recordings of 133 live performances, along with their corresponding audio files. Finally, the month-long practice contributes 101 SC3 scripts from the practice sessions and 43 reflective diaries.

7.1.5.1 Observational datasets

The first structured dataset was used to find relations between systems' characteristics. The emphasis was placed on the gestural interactions of the users with the system. The exact selection criteria can be found in Paper I [35], but the main idea was to find systems that differ in qualitative aspects of gestures. All examples have a video available online.

- **Analytical framework for gestural interactions** [35]
  - 10 systems
    1. Al-jazzari, by Griffiths
    2. Auraglyph, by Salazar
    3. Code LiveCode Live, by Baalman
    4. CodeKlavier CKalcuλator, by Noriega & Veinberg
    5. CodeKlavier hello world, by Noriega & Veinberg

6. iMac Music, by Reus
7. stateLogic Machine, by Diapoulis
8. Threnoscope, by Magnusson
9. TidalCycles (feedforward editor), by McLean
10. Type-A personality, by Collins & Veinberg

The second structured dataset was used in Paper VII [37]. The focus was placed again on the gestural interactions of the users with the system, what I call *interactivity variations*, to denote different expressive manners in the performed gestures. The exact selection criteria can be found in the article, but the main idea was to conduct unstructured observations in the quest for embodiment in live coding. The structured dataset resulted in clustering the systems into three categories, *bottom-up*, *canonical* and *mixed* systems.

- **Interactivity variations** [37]
  - 11 systems
    1. Al-jazzari, by Griffiths
    2. Approximate Programming, by Kiefer
    3. Betablocker, by Griffiths
    4. Code LiveCode Live, by Baalman
    5. CodeKlavier CKalcuλator, by Noriega & Veinberg
    6. iMac Music, by Reus
    7. stateLogic machine, by Diapoulis
    8. superCollider, by Olofsson
    9. Threnoscope, by Magnusson
    10. Using PD, by uiae
    11. Using various, by The Duchess of Turing

The third structured dataset was used in Paper VI [39]. The focus was placed on agent-based systems, and several of the systems were also examined by Xambó [164]. The exact selection criteria can be found in the article, but the main idea was to identify patterns when designing agent-based systems that afford machine listening and machine learning technologies.

- **Designing agent-based systems** [39]
  - 8 systems
    1. Attanayake et al., by Attanayake and colleagues
    2. Autopia, by Lorway
    3. Cacharpo, by Navarro & Ogborn
    4. Flock, by Knotts
    5. Megra, by Reppel
    6. MIRLCa, by Xambó
    7. Ruler, by Paz Ortiz
    8. Wilson et al., by Wilson and colleagues

7.1.5.2 MIR structured dataset

The dataset used for the reproducible study [44] presents possibly the largest dataset used in a live coding study. The original recordings were published on TOPLAP's archive.org web page, and have 133 live coding performances from the Algorave 10 Birthday Party[3]. Each performance has a maximum duration of 10 minutes. The docker container we distribute contains the acoustical feature extraction. In a separate GitHub repository for large files, we distribute the video and audio recordings of each performance. While this is the richest dataset in terms of data (acoustical features, video, audio), the content of it is typically restricted to mostly *standard* live coding performances. Being an algorave, many performers conducted beat-based music, so there is a large influence from specific systems, such as TidalCycles, which is known to facilitate beat-based music making. While our descriptive statistical analysis uses only audio data, the dataset provides opportunities for video analysis. In a manner, we appropriate the TOPLAP's dataset with the intention to facilitate its distribution and present an acoustical analysis of an algorave.

7.1.5.3 Musical practice: Scripts and diaries

The month-long daily practice (Paper IV) [38] contributes a git repository of more than a hundred scripts from live coding sessions and a medium size collection of reflective diaries. The uncommon methodology of the study, employing non-listening conditions, along with the rich data set on code and the experience of writing and conducting the practice session presents unique opportunities for future studies. The previous studies of live coding practice from Collins and Olofsson have already contributed seminal scripts with SC3 code from their daily sessions. Olofsson has conducted similar month-long challenges at least two more times. Thus, the importance of my study is not so much in the produced code and its code quality, but rather in the experimental setup and enriched data set of both the code and the reflective diaries.

7.2 Implications and reflections

7.2.1 Cross-disciplinary boundaries

My research consistently focuses on building bridges between different disciplines in music research with the live coding community. I am specifically interested in communicating knowledge from music psychology and perception communities to live coding. I believe Paper VII [37] can help in this direction, and open a cross-disciplinary discussion that may also be helpful in the comprehension of other electronic music practices. I also hope our reproducible study, presented in Paper V [44], will present a smooth introduction to live coders of the field of MIR studies. Music information retrieval is a highly competitive research field with a prestigious conference (ISMIR) and a competent research community. On the other hand, I often think that ISMIR lacks diversity in research practices,

[3]Algorave 10th Birthday Party: `https://ten.algorave.com` (accessed: 2023-09-05)

with perhaps a too-narrow focus there on quantification and objective measures. The live coding community has a broader and more diverse orientation, and mutual benefits are possible.

7.2.2 Gestural interactions

I use the term *gestural interaction* to discuss the various qualities of gestures within the broad diversity of live coding practices. It seems to be a consensus within the community that gestures differ between traditional instrumental performance and live coding performance, but there has been no attempt to explain the specifics of this difference. Paper VII offers a theoretical understanding, and explains how the term *gestural interaction* is necessary to communicate this. I further elaborate on this topic in Study A and Study B.

It is worth noting that continuous control, by means of direct manipulation using a mouse, has been used to perform program synthesis behaviors in live programming while manipulating HTML DOM file [204]. The connection of advanced levels of liveness with continuous gestures for machine musicianship [40] offers the possibility to address related research topics. Program synthesis could become an important contribution to musical live coding, and the rise of large language models (LLMs) within coding practices might very well become more available in years to come. How such LLMs would look in practice with the use of continuous gestures for musical live coding remains to be seen, but it is my belief that such emergent interactivities would move us away from *gestural control* and closer to *gestural interactions*.

7.2.2.1 Open-loop and closed-loop live coding

One question emerging from this thesis is whether we can engage in musical live coding practices by employing our closed-loop motor program. What would that look like? Instrumental music performance uses both open-loop and closed-loop motor programs, but in live coding there is no such practice, with the exception of systems that use traditional musical instruments. Still, in live coding performances with traditional instruments, the closed loop is only between the performer and the musical instrument (if the instrument affords that), but the live coding system itself is not providing sensory feedback.

The implementation of direct manipulation in the context of bottom-up live coding [40] offers an opportunity to incorporate our closed-loop motor skills. Is this actually possible? Well, we can simply implement an actuator on the knob (i.e., a rotary encoder with a motor), that can resist the imposed directional motion. This can work well, especially in the case of a level-5 liveness setup where the system can indicate a desirable outcome to the user in a tactile manner.

7.2.2.2 The transparency of embodiment

Embodiment is clearly evident in instrumental musical performance, as performers can communicate their expressive intentions to the audience through various bodily movements [140]. In contrast, in live coding, movement is

minimal – though it is transparent. In several studies, I was able to infer the mode of interaction and how musicians move by simply observing, mostly by viewing screen recordings from videos online. It is interesting that we cannot really "hide" our bodily engagement with an interface even when the audience sees only the performance interface and not the movements of the performers themselves.

7.2.3 Liveness and code-preview

The implications for liveness in the work presented here relate to not only what can be learned when designing agent-based systems, but also how code previews can influence our practices when using bottom-up systems.

The conceptual framework presented in Paper VI, is not informative on what components can be used for Level-5 and Level-6 liveness (Figure 5.3). This is because among the observed use cases, there is no system that claims to be a *tactically predictive* system (i.e., level-5 liveness). This is understandable, as any predictive model that generates a novel code chunk or pattern for the coder does not necessarily informs the user what to do next in a musical live coding setting. This is something that may be possible after extensive training and system personalization, but I have not seen anything of this nature yet.

Code previews are not widely used in live performances. I only recently saw a live setup by Lizzie Wilson[4] that employed code previews[5] during a live performance, and there are a few demos online that demonstrate various code preview features [207], [208]. Roberts and Wakefield [104] discussed how audio previewing can be temporally demanding in practice and that code previewing also is cognitively demanding. So, it is likely that live coders do not want to take such risks during a performance. But what could it mean in practice to use preview code during a canonical live coding session? Doing so would mean that the system could either *mind-guess* the performer's plans or make some informed decision about future versions of the running program. The case of mind-guessing is what Tanimoto [43] refers to as a *strategic prediction* (level-6 liveness). If the system could inform the user about possible alternative versions of the written code then it would be a *tactically predictive* system (level-5 liveness). The next step beyond mind-guessing would inevitably be *mind-reading* the user's intentions, and it may not be that far away in the future, as brain decoding for image [209] and music reconstruction [210] are getting more and more robust using large-scale learning models. Such applications are not likely to happen soon within conventional programming practices, as they lack any multimodal engagement between the semantics of the code and the user's experience.

7.2.4 Agency: Relational, shared and influential

Agency is related to sensorimotor prediction [31]. The liveness(-es) of the cybernetic systems I have discussed that exhibit agency, cause the assignment

[4]Lizzie Wilson Research Page, `https://lwlsn.github.io/`

[5]During the Algorave as part of the AIMC 2023 conference in Brighton, UK.

of authorship between the human and the machine to become blurred. If the closed-loop system proposed above were to be implemented, then that would certainly demonstrate some *shared agency* between the human and the system. I use here the term *shared agency* to indicate *joint actions* between the human and the system that would have a physical manifestation exhibited, such as as an actual force on a potentiometer. Such a setup would exhibit *mutual adaptations* between the parties involved. The system would take an actual co-creative role, by facilitating or opposing the coders' intentional gestures with a knob, say. For instance, imagine a scenario where the system opposes the coders intention to generate a particular token – something like a game. In this case, the system would trigger a response to the rotary encoder (equipped with a motor) to resist the clockwise turn when the coder attempts to move it clockwise, and the system would offer no resistance if the coder turns the knob counter-clockwise.

The suggested implementation presented above moves beyond the *relational* aspects of agency in live coding [49]. As a vision, I wonder how this influential aspect of agency would influence various practitioners. It looks to me that a closed-loop system might be highly personalized. Of course, most humans roughly share the same physiological characteristics, but how each one of us would perceive shared agency with computing machinery, might, I think, widely differ.

Chapter 8

Conclusions

I started this article-based thesis by presenting the contributing articles in three categories: *applied*, *conceptual* and *methodological* contributions. Whereas the boundaries between the categories are sometimes blurred and blend with one another, this framing presents distinctly different ways of knowing, different *paths* we follow in the process of learning and examining the unknown. Live coding is a fairly new term, as it was coined 20 years ago [22]. Whereas it has become an established research field within computer music and music technology in general, it is largely unknown to the general academic audiences. Magnusson [48] has discussed the question of whether the term *live coding* will persist, or whether it is a temporary designation. Besides, it is also an open research field, that resists the imposition of strict boundaries and interdisciplinary norms. I hope that the field can keep its fresh and playful character for many years to come.

My main intention throughout the thesis is to understand how we live code for music performance practices, and then to explain it to others. In most articles I focus on observing how other live coders practice, and on how they develop their systems. Obviously, the target audience is mainly the live coding community, as has been explicitly stated in most of the articles. I am always thankful and happy to see how this community is changing, progressing and evolving.

The home of the live coding community, in terms of publishing, is the ICLC. Also, the related online annual meeting on *Hybrid Live Coding Interfaces* aims to improve the inclusivity, accessibility, and diversity of the community and its practices. Besides these obvious connections, the thesis may prove to be of use to related research forums, mainly the NIME community and all computer music-related fora, such as ICMC, SMC, AIMC, and more. Finally, the broader audience of this thesis may also include readers from the communities for tangible interaction (TEI) and human-computer interaction (CHI), as the main focus of the thesis is on expressive gestural interactions. Live coding has been presented also in prestigious ACM conferences, in particular Creativity & Cognition [211], [212] and CHI [213], [214]. Music is an ideal context in which to study bodily movement and temporally-dependent interactions, and live

coding in particular exemplifies how processes unfold and transform in time.

8.1 Open problems in live coding

There is a lack of understanding of live coding practices in terms of their perceptual and psychological implications. Only a few articles explicitly target this area, and there is very little writing reflecting upon the implications of live coding practices. I contributed a study [37] that reflects on this practice, and presents a basic discussion from the perspective of music performance for live coding. This thesis contributes a study of live coding in relation to music psychology and music perception, and addresses issues of expressive performance and musical interaction (i.e., the nature of our bodily engagement in live coding).

In that respect, this thesis provides a new point of view, as in the live coding literature sometimes we have tended to put more attention on the system side and ignore the human side. In this way, this work sheds light on several issues related to human perception and cognition, particularly topics related to music listening, musical imagery, memory, and sensorimotor control. It does not directly address anything related to emotional resonance and aesthetic enjoyment, but I hope to motivate others to contribute to this fairly unexplored field in live coding. The study of musical interaction in live coding: that is, how we entrain with its running processes, is still in its infancy, and we essentially do not yet know how to approach this topic. It may be impossible at this point to comprehend musical entrainment in live coding vis-a-vis its nature in traditional musicianship, but I would argue that it is a journey worth the risk. On the topic of expression, there are several important contributions, like the sections on Code LiveCode Live, CodeKlavier CKalcuλator, the stateLogic machine, and Approximate Programming. Different individuals have different ways of mapping expressive interactions during a performance, and I present a discussion that explores these relations as "mental models" between code and sound.

One thing observed across all the studies is that live coding is a diverse topic that cannot fit into a neatly defined box. This is certainly not a problem, but a rewarding observation, that live coding persists in its wildness [48]. The generativity of code affords such a rich range of possibilities, that no system closely resembles any other in its details, and only very high-level generalizations can be made about their correspondences. Even such high-level abstractions are often not adequate to describe certain live coding systems, which asserts the generativity of them, and of live coding generally. This observation is an optimistic view, for the community as it shows that live coding is not a saturated research field. I would say in fact that we are now on the threshold of the most fruitful period of live coding.

After 10 years of live coding, there have been quite a few efforts to deliver solid and reliable tools to the community. Numerous languages have been developed, and now we are entering into the era where many individuals develop personalized languages and libraries for machine learning during performance.

From now on, we will likely enter a lively experimentation era with AI[1]. The first personalized systems have already come out, and collaborative efforts have delivered powerful (eco-)systems for employing AI in live coding. How large language models will impact the field is unknown and unpredictable, but I would expect to see advanced levels of liveness (such as using code previews) more often. It remains to be seen how and if liveness will also impacts audio generation. Issues of AI ethics and security also are sure to impact live coding and the community should be attentive to these risks.

8.2 Research outcomes

The driving motivation behind this thesis was to explore how we can live code in a manner that feels more like instrumental musicianship. The main research question behind this thesis was to ask what happens when we employ subconscious modes of cognition while live coding. This is a long-standing question which has been under discussion since 2014 [42]. All other research questions that I engage with came out in the process of doing this Ph.D. research. Some long-standing questions I have had, such as how we practice live coding, and how we gesture with generative algorithms, have been in the back of my mind through the years, but I had never really conducted a systematic effort to address them before.

8.2.1 Answers to the research questions

- RQ1: Are pre-reflective processes evident during a musical live coding performance?
  - What are the necessary conditions?
  - In which cases do these conditions occur?

  Pre-reflective processes, also known as fast or subconscious processes, are essential in traditional musicianship, as they are partly responsible for flow and groove states. In live coding the importance of these matters is is not obvious, as we are engaged in writing a computer program, which itself is essentially a logical task highly demanding on attentional resources and high-level cognitive functions, like reasoning and planning. But there are some obvious instances where fast processes are employed in musical live coding, like in the CodeKlavier system where the coder is playing the piano (also known as piano-coder). Besides pointing out these obvious connections, I presented in [37] a study that mainly focuses on answering this question. I identified several more instances based on the fast time responses, such as short-edits, typing automaticity, bottom-up systems, and systems that afford direct manipulation. Also, minimalistic systems, like ixi-lang and TidalCycles – systems that exhibit elegant and

[1]The closing keynote ICLC 2023 by the reincarnation of Click Nilson was performed with ChatGPT as a main tool. `https://www.youtube.com/live/FSBtvtxP008?si=ALTKerLzZPmtn7Ez&t=9167` (accessed 2023-08-29)

short code expressions – are seen as our best chance to accommodate fast processes in so-called *canonical live coding*. For fast processes to be evident, it is necessary that we be engaged in a bodily manner during the music-making. This can happen in different ways, but one obvious manner is the user is required to respond promptly within less than a few seconds, so that we must engage our sensorimotor system. Instances that facilitate bodily coordination (i.e., bodily entrainment), are particularly important, and may be the missing link between high-level motor functions, such as planning, and low-level motor functions, such as sensorimotor control.

- RQ2: How can generative algorithms influence musical gestures used during live coding performance?
  - How is this evident in live coding, and how is its appearance similar to or different from its appearance in work with digital musical instruments and other interactive music systems?

  How we gesture in machine musicianship radically differs how we gesture in instrumental musicianship. There is a significant difference, because unlike in instrumental musicianship, most expressions of bodily gestures in live coding are *not* sound-producing. This thesis presents a series of articles focusing on observing and theorizing on gestures [35]–[37], also presenting a software prototype that implements direct manipulation [40] for bottom-up live coding practices. I argue that how we gesture with algorithms in bottom-up live coding differs from the *standard paradigm*, as every step we make modifies, to some extent, the running algorithms. I support the term *gestural interaction* to denote this enactment between the human and system sides. I also see qualitative differences between gesturing in live coding and in DMIs or other IMS, as we typically aim for goal-directedness to monitor how coding structures unfold in time. When a live system is not supporting monitoring of the code, then the gestural interactions may feel more like performing with generative algorithms used in DMIs and IMS.

- RQ3: What creativity support technologies are used in musical live coding?
  - What is the role of liveness, and how we can use the technological advancements of liveness?

  Technologies for creativity support in musical live coding practices are expressed either as text, sound, or visualization. Two studies were conducted to directly address creativity issues [38], [39], and one study directly addresses issues of liveness [40]. Visualizations are used extensively in live coding, and if not only for aesthetic utilization, their main function is to *support creativity* mechanisms. Here I argue for the importance of descriptive notation as it may influence our embodied percepts. I also argue that simple visual prompts can be useful as presented with a GUI helper and can act either as reminders (e.g., "forgot panning")

or as prompts that can creatively influence our practices. I note that liveness is underused in live coding, likely because it can demand increased mental effort during a performance. To date, the most common expression of advanced levels of liveness in live coding practices is related to code preview. I suggest that bottom-up systems can be an interesting approach to "revisit" our interest in advanced levels of liveness. Maybe small independent programs with specialized functions can also bring forth emergent interactivities for advanced levels of liveness. Being perhaps interconnected in a network, such small programs may be seen as *entangled minimal agents*.

- RQ4: How do we practice musical live coding with assistive technologies?

  - What is the role of multimodal experience in live coding practice?

  There is no unique and easy way to practice live coding, so I am unclear whether assistive technologies can "fast-track" our learning process. If the reader wonders how to live code, the only advice is to practice at length, similar to what one does in traditional musicianship. I found out that my skills dramatically improved after a month-long daily practice. Meta-learning practices, such as reflective diaries, can be helpful. Visual cues, such as the spectral plots, sound level meters or simple reminder prompts can be helpful, although I would expect that preferences in the use of such tools may be highly individualistic. Also, I found it fascinating that the defamiliarization that occurs when not listening to the output can be used as a creative technique. It became evident that when non-listening to the musical outcome various tasks become impossible, such as musical transitions and perception of rhythm. Finally, assistive technologies, when first added to one's work flow, may indeed impede concentration and focus during reasoning and abstract thought, but they eventually move to the background of our attention.

- RQ5: How can and do we design agent-based systems for musical live coding?

  This is the main motivation for Paper VI [39], where it became clear that there is little attention paid to machine listening in the live coding community. There is a tendency to apply predictive models to the code but not to the generated musical outcome. Feedback mechanisms could be useful to this end, but also seem to have a small influence on current live coding practices. Also, while most systems apply learning to some sort of textual data, either to code or numerical parameters, there is little focus on text generation and program synthesis, therefore having little to no influence on the prescriptive part of the notation. I think this tendency is likely to change as large language models becoming more and more accessible. Finally, advanced levels of liveness (such as *tactically predictive* and *strategically predictive*, according to Tanimoto's hierarchy of liveness [43]; see section 5.2.1) seem to be out of scope for current

practitioners, because of the required cognitive load and lack of adecquate tools.

- RQ6: How is risk expressed in musical live coding?

  Risk is multifaceted in live coding, it addresses issues related to practice where we typically tend to minimize risks during a performance, but extends beyond the musical practice and can also address ethical and cultural aspects. Here, I focus on musical practice and I identify that almost all bottom-up systems do not run any significant risk of crashing. I suggest this state of affairs should be revisited and I see the emergent role of gestural interactions as introducing a desirable level of risk in bottom-up practices.

8.2.2 Limitations

> "Question everything apart from this statement."
>
> Click Nilson, last slide on the ICLC 2023 closing keynote.

I start this section by explaining the self-reference that is the first word of this sentence. I wrote this thesis in the first-person to express its subjective nature. This subjectivity relates to various research designs and methods, such as the observations I conducted in four articles, the reflective diaries and more. One important note about my *personal language* is that I also introduce terminologies to explain my research outcomes. This applies throughout, from the title of this dissertation where I use the term "gestural interaction" to the use of the term "bottom-up" for some live coding systems. The self-referential manner I started this paragraph with is inspired by the opening quote for this section from Click Nilson[2]. Self-reference is related to paradoxes and live coding is certainly a paradoxical practice. In computation, paradoxes relate to undecidability [25, p. 225] and live coding is essentially on-the-fly decision-making [48].

One obvious limitation of the bottom-up system presented in this thesis, such as the stateLogic machine and its follow up versions, is that these systems are not Turing complete. So the first issue arising is determining how useful these systems are in the first place. It is possible to employ imaginary instruction sets, like in the TOPLAP app and Betablocker, by mapping the chosen set of tokens to different instructions. Yet, even this would not solve the known issue of bottom-up systems on being not risky enough. Maybe, the path to be taken for bottom-up systems should be to make them Turing complete and introduce risk by the presence of halting states, and possible errors in parsing rules. Of course that would impact a performance and may be unpleasant to wait for one to two minutes for a code modification. At the same time, in current conventional live coding the performer may still be required to type for couple of minutes to make a change. If bottom-up systems were ever to

[2]Click Nilson, a persona of Nick Collins, has authored academic articles, musical performances, and public presentations.

become mature, then the development of more elegant and musical systems in years to come would become more possible.

The conceptual frameworks presented in this manuscript adhere to the present live coding "state of mind" and are not meant to be treated as "set in stone." The conceptual framework introduced in Paper I and Paper II has some limitations as it can be hard to map certain systems explicitly (i.e., using a single arrow). Further research along these lines would have to look more thoroughly at concepts such as the complexity of the interface and the precision of chosen mappings, and how these concepts may relate to our gestural interactions. The conceptual framework presented in Paper VI presents a paradigm more open to interpretation due to its modular design. The hard question to answer is whether such a conceptual framework can be of use to the live coding practitioner. This is a question I cannot answer.

Does the thesis opens a pandora box with the *bottom-up* and *top-down* terminologies [48]? It has been remarked that *tensions and techniques* may influence each other [104]. I mainly propose that a clearer description of the systems used would be beneficial, and I highlight some characteristics of bottom-up systems that differ from the standard approach to live code.

This thesis barely touches upon issues of visual perception and crossmodal interactions. Each of these topics requires a rigorous undertaking, and as far as I know there are no previous studies that address these topics at length. Visual perception is particularly important for visual live coding, but also important for musical live coding. Crossmodal interactions, how one modality interacts with another, is relevant to live coding as a whole and touches upon issues related to time perception and sequential actions. It would be particularly important to musical live coding to study how auditory and motion perception interact with each other, and how audition interacts with vision.

Finally, as mentioned also in the introduction, this thesis does not cover topics related to collaborative and networked live coding, ethics, or gender studies. Collaborative and networked live coding is central to the live coding community and there are several tools available since the early days. I think that in the near future more tools will be developed to enhance accessibility for telematic performances. Collaborative editing is definitely something that we are going to see more often, and such work has already been implemented in the flok editor (`https://flok.cc/`) and in Estuary (`https://estuary.mcmaster.ca/`). Ethical studies are more topical than ever in live coding, especially with the advent of large language models, and gender studies are mainly addressed by the feminist live coding societies, like LivecoderA[3].

8.3 Future directions in musical live coding

I start the future directions section by pointing out what would be a next step for this thesis and then address some of the possible future directions of musical live coding.

[3] `https://livecodera.glitch.me/`

The immediate application that I see as worth experimenting with is to develop a bottom-up system that employs the closed-loop motor program. I suggested here that this can be done using *rotary force-feedback* mechanisms, by means of rotary encoders equipped with motors. Rotary force-feedback is used in novel musical interfaces [151], [215], but what I propose here is to incorporate our closed-loop motor skills to predict programming behaviours. This is an effective means to get physical haptic feedback from the system to the coder. Usually, we use this interactivity the other way around, to get decisions made by the human to the machine [216]. Here, I propose to do this bidirectionally, to have system decisions affect the nature of the human's interface, and to go a step further by exploring the machine as an agent that affects future previews of the system. That would open philosophical questions on the goal-directedness of live coding and human-machine joint actions. The latter development opens a path for human-agent as well as human-human inter-subjective experience in live coding, and I hope to see more focus on this topic in the literature.

Finally, as presented by Click Nilson in his closing keynote, the future of live coding will be widely impact by large language models. The research and practice field is largely unexplored, and only a few live coders have been experimenting with these types of interactivities as we speak. I see great potential in these practices and I also see that bottom-up systems can benefit from the adoption of such practices.

I do not know how live coding will look ten years from now. My speculative vision would be to give more space to practices that affords *proscriptive* evaluations to take place, that is to enable programmable actions that are *good enough*. Maybe such systems coupled with an inter-subjective view on cognition would bring forth emergent qualities of live coding practices that are unforeseen. I would also hope that live coding would contribute to new ways of expression for writing systems [217], and particularly to make tightly coupled connections between different modalities, such as audition, vision and tactile percepts.

Bibliography

[1] R. Rowe, *Machine musicianship*. MIT press, 2004 (cit. on pp. 2, 38, 41).

[2] R. Rowe, *Interactive music systems: machine listening and composing*. MIT press, 1992 (cit. on pp. 2, 14, 42, 57).

[3] A. R. Jensenius and M. J. Lyons, *A nime reader: Fifteen years of new interfaces for musical expression*. Springer, 2017, vol. 3 (cit. on p. 3).

[4] B. Verplank, M. Gurevich and M. Mathews, "2002: The plank: Designing a simple haptic controller," *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 59–70, 2017 (cit. on p. 3).

[5] M. J. Lyons, M. Hähnel and N. Tetsutani, "2003: Designing, playing, and performing with a vision-based mouth interface," *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 107–124, 2017 (cit. on p. 3).

[6] F. Bevilacqua, F. Guédy, N. Schnell, E. Fléty and N. Leroy, "Wireless sensor interface and gesture-follower for music pedagogy," in *Proceedings of the 7th international conference on New interfaces for musical expression*, 2007, pp. 124–129 (cit. on p. 3).

[7] A. McPherson, "2012: Touchkeys: Capacitive multi-touch sensing on a physical keyboard," *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 419–432, 2017 (cit. on p. 3).

[8] M. Wright, A. Freed and A. Momeni, "2003: Opensound control: State of the art 2003," *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 125–145, 2017 (cit. on p. 3).

[9] E. Berdahl and W. Ju, "Satellite ccrma: A musical interaction and sound synthesis platform.," in *NIME*, 2011, pp. 173–178 (cit. on p. 3).

[10] D. Wessel and M. Wright, "Problems and prospects for intimate musical control of computers," *Computer music journal*, vol. 26, no. 3, pp. 11–22, 2002 (cit. on pp. 3, 42).

[11] A. Hunt, M. M. Wanderley and M. Paradis, "The importance of parameter mapping in electronic instrument design," *Journal of New Music Research*, vol. 32, no. 4, pp. 429–440, 2003 (cit. on p. 3).

[12] S. Jordà, "Sonigraphical instruments: From fmol to the reactable.," in *NIME*, vol. 3, 2003, pp. 70–76 (cit. on p. 3).

[13] D. Birnbaum, R. Fiebrink, J. Malloch and M. M. Wanderley, "Towards a dimension space for musical devices," in *NIME*, vol. 5, 2005, pp. 192–195 (cit. on pp. 3, 50).

[14] A. R. Jensenius, "2014: To gesture or not? an analysis of terminology in nime proceedings 2001–2013," *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 451–464, 2017 (cit. on p. 3).

[15] P. Cook, "2001: Principles for designing computer music controllers," *A NIME Reader: Fifteen years of new interfaces for musical expression*, pp. 1–13, 2017 (cit. on p. 3).

[16] S. Jordà, "2003: Sonigraphical instruments: From fmol to the reactable," *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 89–106, 2017 (cit. on p. 3).

[17] S. O'Modhrain and G. Essl, "Pebblebox and crumblebag: Tactile interfaces for granular synthesis," in *Proceedings of the 2004 conference on New interfaces for musical expression*, 2004, pp. 74–79 (cit. on p. 3).

[18] G. Wang and P. R. Cook, "On-the-fly programming: Using code as an expressive musical instrument.," in *NIME*, vol. 4, 2004, pp. 138–143 (cit. on pp. 3, 21).

[19] R. Fiebrink, G. Wang and P. R. Cook, "2007: Don't forget the laptop: Using native input capabilities for expressive musical control," *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 285–297, 2017 (cit. on p. 3).

[20] A. Freed, "The fingerphone: A case study of sustainable instrument redesign.," in *NIME*, 2012 (cit. on p. 3).

[21] C. Roberts, G. Wakefield and M. Wright, "2013: The web browser as synthesizer and interface," *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 433–450, 2017 (cit. on p. 3).

[22] N. Collins, A. McLean, J. Rohrhuber and A. Ward, "Live coding in laptop performance," *Organised sound*, vol. 8, no. 3, pp. 321–330, 2003 (cit. on pp. 3, 6, 15, 21, 55, 71, 95).

[23] J. Rohrhuber, A. de Campo and R. Wieser, "Algorithms today notes on language design for just in time programming," in *International Computer Music Conference*, 2005, p. 291 (cit. on pp. 3, 18, 21).

[24] J. Rohrhuber and A. De Campo, "Improvising formalisation: Conversational programming and live coding," *New Computational Paradigms for Computer Music. Delatour France/Ircam-Centre Pompidou*, 2009 (cit. on p. 3).

[25] A. F. Blackwell, E. Cocker, G. Cox, A. McLean and T. Magnusson, *Live coding: a user's manual*. MIT Press, 2022 (cit. on pp. 3, 4, 16, 18, 22, 26, 47, 60, 71, 100).

[26] E. Cocker, "Performing thinking in action: The meletē of live coding," *International Journal of Performance Arts and Digital Media*, vol. 12, no. 2, pp. 102–116, 2016 (cit. on pp. 3, 4, 72).

[27] J. Armitage, "Spaces to fail in: Negotiating gender, community and technology in algorave.," *Dancecult: Journal of Electronic Dance Music Culture*, vol. 10, no. 1, 2018 (cit. on pp. 4, 10).

[28] S. Knotts, "Live coding and failure," *The Aesthetics of Imperfection in Music and the Arts: Spontaneity, Flaws and the Unfinished*, p. 189, 2020 (cit. on pp. 4, 71, 74, 75).

[29] F. Capra and P. L. Luisi, *The systems view of life: A unifying vision*. Cambridge University Press, 2014 (cit. on p. 4).

[30] M. Leman, *Embodied music cognition and mediation technology*. MIT press, 2007 (cit. on pp. 4, 7, 41).

[31] M. Leman, *The expressive moment: How interaction (with music) shapes human empowerment*. MIT press, 2016 (cit. on pp. 4, 7, 41, 48, 81, 92).

[32] H. R. Maturana and F. J. Varela, *The tree of knowledge: The biological roots of human understanding*. New Science Library/Shambhala Publications, 1987 (cit. on pp. 4, 42).

[33] F. J. Varela, E. Thompson and E. Rosch, *The embodied mind, revised edition: Cognitive science and human experience*. MIT press, 2017 (cit. on pp. 4, 41, 42, 88).

[34] S. Ljungblad, "Beyond users: Grounding technology in experience," Ph.D. dissertation, Institutionen för data-och systemvetenskap (tills m KTH), 2008 (cit. on p. 5).

[35] G. Diapoulis and P. Dahlstedt, "An analytical framework for musical live coding systems based on gestural interactions in performance practices," in *Internation Conference on Live Coding*, Valdivia, Chile, 2021 (cit. on pp. 6, 32, 43, 48, 49, 51, 68, 82, 83, 87, 88, 98).

[36] G. Diapoulis and P. Dahlstedt, "The creative act of live coding practice in music performance.," in *PPIG*, 2021 (cit. on pp. 6, 43, 49, 68, 82, 83, 98).

[37] G. Diapoulis, "Musical live coding in relation to interactivity variations," *Organised Sound*, 1–13, 2023. DOI: 10.1017/S1355771823000444 (cit. on pp. 6, 17, 22, 30, 43, 44, 47, 54, 56, 75, 80, 82, 83, 85, 86, 89, 90, 96–98).

[38] G. Diapoulis, "Livecode me: Live coding practice and multimodal experience," in *Proceedings of Psychology of Programming Interest Group*, PPIG2022, 2022 (cit. on pp. 6, 16, 17, 62, 67, 69, 77, 80, 87, 90, 98).

[39] G. Diapoulis, "Liveness and machine listening in agent-based systems for musical live coding," in *Proceedings of AI Musical Creativity*, AIMC, 2023 (cit. on pp. 6, 65, 67–69, 88, 89, 98, 99).

[40] G. Diapoulis, I. Zannos, K. Tatar and P. Dahlstedt, "Bottom-up live coding: Analysis of continuous interactions towards predicting programming behaviours," *NIME*, 2022 (cit. on pp. 6, 23, 27, 31–34, 36–38, 66, 69, 80, 83, 91, 98).

[41] G. Diapoulis and I. Zannos, "A minimal interface for live hardware coding," *Live Interfaces*, 2012 (cit. on pp. 6, 25, 27, 28, 36, 43, 83).

[42] G. Diapoulis and I. Zannos, "Tangibility and low-level live coding," in *ICMC*, 2014 (cit. on pp. 6, 19, 27, 31–33, 35, 36, 83, 97).

[43] S. L. Tanimoto, "A perspective on the evolution of live programming," in *2013 1st International Workshop on Live Programming (LIVE)*, IEEE, 2013, pp. 31–34 (cit. on pp. 6, 21, 22, 30, 58–60, 66, 92, 99).

[44] G. Diapoulis and M. Carlé, "Reproducible musical analysis of live coding performances using information retrieval: A case study on the algorave 10th anniversary," in *International Conference on Live Coding*, ICLC, 2023 (cit. on pp. 6, 64, 68, 69, 75, 78, 80, 82, 87, 90).

[45] T. Magnusson and E. H. Mendieta, "The acoustic, the digital and the body: A survey on musical instruments," in *Proceedings of the 7th international conference on New interfaces for musical expression*, 2007, pp. 94–99 (cit. on p. 7).

[46] M. Merleau-Ponty, D. Landes, T. Carman and C. Lefort, *Phenomenology of perception*. Routledge, 2013 (cit. on p. 9).

[47] J.-P. Banâtre, P. Fradet, J.-L. Giavitto and O. Michel, *Unconventional Programming Paradigms: International Workshop UPP 2004, Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers*. Springer Science & Business Media, 2005, vol. 3566 (cit. on p. 9).

[48] T. Magnusson, "Herding cats: Observing live coding in the wild," *Computer Music Journal*, vol. 38, no. 1, pp. 8–16, 2014 (cit. on pp. 9, 26, 84, 95, 96, 100, 101).

[49] A. R. Brown, "Performing with the other: The relationship of musician and machine in live coding," *International Journal of Performance Arts and Digital Media*, vol. 12, no. 2, pp. 179–186, 2016 (cit. on pp. 9, 18, 93).

[50] S. Aaron, "Sonic pi–performance in education, technology and art," *International Journal of Performance Arts and Digital Media*, vol. 12, no. 2, pp. 171–178, 2016 (cit. on p. 10).

[51] J. Freeman, B. Magerko, D. Edwards, T. Mcklin, T. Lee and R. Moore, "Earsketch: Engaging broad populations in computing through music," *Communications of the ACM*, vol. 62, no. 9, pp. 78–85, 2019 (cit. on p. 10).

[52] A. Selvaraj, E. Zhang, L. Porter and A. G. Soosai Raj, "Live coding: A review of the literature," in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, 2021, pp. 164–170 (cit. on p. 10).

[53] A. R. Jensenius, *Sound Actions: Conceptualizing Musical Instruments*. MIT Press, 2022 (cit. on pp. 10, 42, 47, 73, 86).

[54] C. Nilson, "Live coding practice," in *Proceedings of the 7th international conference on New interfaces for musical expression*, 2007, pp. 112–117 (cit. on pp. 10, 16, 17, 44, 47, 62).

[55] T. Magnusson, "Scoring with code: Composing with algorithmic notation," *Organised Sound*, vol. 19, no. 3, pp. 268–275, 2014 (cit. on pp. 10, 17).

[56] P. Dahlstedt, *Sounds Unheard of: Evolutionary algorithms as creative tools for the contemporary composer*. Chalmers Tekniska Högskola, 2004 (cit. on pp. 10, 42, 57, 58, 80).

[57] C. Haworth, "Technology, creativity, and the social in algorithmic music," in *The Oxford Handbook of Algorithmic Music*, A. McLean and R. T. Dean, Eds., Oxford University Press, 2018, ch. 31, pp. 557–581 (cit. on pp. 10, 12, 55).

[58] S. Gresham-Lancaster, "The aesthetics and history of the hub: The effects of changing technology on network computer music," *Leonardo Music Journal*, vol. 8, no. 1, pp. 39–44, 1998 (cit. on p. 11).

[59] A. McLean, *Hacking perl in nightclubs*, 2004. [Online]. Available: `https://www.perl.com/pub/2004/08/31/livecode.html/` (cit. on pp. 12, 21).

[60] N. Collins, "Origins of algorithmic thinking in music," in *The Oxford Handbook of Algorithmic Music*, A. McLean and R. T. Dean, Eds., Oxford University Press, 2018, ch. 4, pp. 67–78 (cit. on p. 12).

[61] T. Magnusson, "Algorithms as scores: Coding live music," *Leonardo Music Journal*, vol. 21, pp. 19–23, 2011 (cit. on pp. 12, 19, 87).

[62] M. Carruthers and J. M. Ziolkowski, *The Medieval Craft of Memory: An anthology of texts and pictures*. University of Pennsylvania Press, 2002 (cit. on p. 12).

[63] L. A. Hiller, "Computer music," *Scientific American*, vol. 201, no. 6, pp. 109–121, 1959 (cit. on p. 13).

[64] C. Matthews, *Algorithmic thinking and central javanese gamelan.* 2018 (cit. on p. 13).

[65] F. P. Brooks, A. Hopkins, P. G. Neumann and W. V. Wright, "An experiment in musical composition," *IRE Transactions on Electronic Computers*, no. 3, pp. 175–182, 1957 (cit. on p. 13).

[66] I. Xenakis, *Formalized music: thought and mathematics in composition.* Pendragon Press, 1992 (cit. on p. 13).

[67] J. Bischoff, R. Gold and J. Horton, "Music for an interactive network of microcomputers," *Computer Music Journal*, pp. 24–29, 1978 (cit. on p. 13).

[68] C. Brown and J. Bischoff, "Indigenous to the net: Early network music bands in the san francisco bay area," *Available at crossfade. walkerart.org/brownbischoff*, 2002 (cit. on p. 13).

[69] T. Perkis, *Hub music*, Cassette booklet, 1987 (cit. on p. 13).

[70] C. Scholz, "The scores," in *The Hub: Pioneers of Network Music*, L. Brümmer, C. Preiß and G. Robair, Eds., Heidelberg: Kehrer Verlag, ZKM, 2021, pp. 71–125 (cit. on p. 13).

[71] W. F. Thompson, *Music, thought, and feeling: Understanding the psychology of music*. Oxford university press, 2015 (cit. on pp. 13, 42).

[72] D. Zicarelli, "M and jam factory," *Computer Music Journal*, vol. 11, no. 4, pp. 13–29, 1987 (cit. on p. 14).

[73] P. Dahlstedt, "A mutasynth in parameter space: Interactive composition through evolution," *Organised Sound*, vol. 6, no. 2, pp. 121–124, 2001 (cit. on pp. 14, 34).

[74] C. Roads and M. Mathews, "Interview with max mathews," *Computer Music Journal*, vol. 4, no. 4, pp. 15–22, 1980 (cit. on p. 14).

[75] J. McCartney, "Rethinking the computer music language: Super collider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, 2002 (cit. on p. 15).

[76] N. M. Collins, "Towards autonomous agents for live computer music: Realtime machine listening and interactive music systems," Ph.D. dissertation, Citeseer, 2007 (cit. on pp. 15, 57, 58).

[77] A. F. Blackwell, "Coding or ai? tools for control, surprise and creativity.," in *PPIG*, 2022, pp. 57–66 (cit. on p. 15).

[78] J. Rohrhuber, A. de Campo, R. Wieser, J.-K. Van Kampen, E. Ho and H. Hölzl, "Purloined letters and distributed persons," in *Music in the Global Village Conference (Budapest)*, 2007 (cit. on p. 15).

[79] N. Collins, *Handmade electronic music: the art of hardware hacking*. Taylor & Francis, 2020 (cit. on p. 15).

[80] M. Van Atten, "Brouwer, as never read by husserl," *Synthese*, vol. 137, no. 1-2, pp. 3–19, 2003 (cit. on p. 16).

[81] M. Van Atten, *Brouwer meets Husserl: on the phenomenology of choice sequences*. Springer Science & Business Media, 2006, vol. 335 (cit. on p. 16).

[82] K. Jakubowski, "13 musical imagery," *The Cambridge handbook of the imagination*, p. 187, 2020 (cit. on p. 17).

[83] W. Brodsky, Y. Kessler, B.-S. Rubinstein, J. Ginsborg and A. Henik, "The mental representation of music notation: Notational audiation.," *Journal of Experimental Psychology: Human Perception and Performance*, vol. 34, no. 2, p. 427, 2008 (cit. on p. 17).

[84] P. E. Keller, "Mental imagery in music performance: Underlying mechanisms and potential benefits," *Annals of the New York Academy of Sciences*, vol. 1252, no. 1, pp. 206–213, 2012 (cit. on p. 17).

[85] P. E. Keller and I. Koch, "Action planning in sequential skills: Relations to music performance," *Quarterly Journal of Experimental Psychology*, vol. 61, no. 2, pp. 275–291, 2008 (cit. on p. 17).

[86] C. Palmer, "Music performance," *Annual review of psychology*, vol. 48, no. 1, pp. 115–138, 1997 (cit. on pp. 17, 42, 54, 81, 82).

[87] S. Emmerson, *Living electronic music*. Routledge, 2017 (cit. on pp. 17, 59).

[88] S. P. Gill, *Tacit engagement*. Springer, 2015 (cit. on p. 18).

[89] P. Dahlstedt, "Musicking with algorithms: Thoughts on artificial intelligence, creativity, and agency," in *Handbook of Artificial Intelligence for Music*, Springer, 2021, pp. 873–914 (cit. on pp. 18, 19, 35, 57).

[90] A. McPherson and K. Tahıroğlu, "Idiomatic patterns and aesthetic influence in computer music languages," *Organised sound*, vol. 25, no. 1, pp. 53–63, 2020 (cit. on p. 18).

[91] T. Magnusson, *Sonic writing: technologies of material, symbolic, and signal inscriptions*. Bloomsbury Publishing USA, 2019 (cit. on p. 19).

[92] T. Magnusson, "Of epistemic tools: Musical instruments as cognitive extensions," *Organised Sound*, vol. 14, no. 2, pp. 168–176, 2009 (cit. on p. 19).

[93] N. Collins, *Semiconducting: Making music after the transistor.* 2013 (cit. on p. 19).

[94] G. Lakoff and M. Johnson, "Metaphors we live by. chicago: Univ," *Press, Chicago/IL*, 1980 (cit. on pp. 19, 42).

[95] F. Olofsson, "Cybernetic music in practice," *Ideas Sonicas*, vol. 23, pp. 37–40, 2020 (cit. on pp. 21, 73).

[96] P. Dahlstedt, "Action and perception: Embodying algorithms and the extended mind," in *The Oxford Handbook of Algorithmic Music*, A. McLean and R. T. Dean, Eds., Oxford University Press, 2018, ch. 3, pp. 41–65 (cit. on pp. 21, 46).

[97] A. Sorensen, "Impromptu: An interactive programming environment for composition and performance," in *Australasian Computer Music Conference 2009: Improvise*, 2005 (cit. on p. 21).

[98] A. S. Bregman, *Auditory scene analysis: The perceptual organization of sound*. MIT press, 1994 (cit. on p. 22).

[99] B. Snyder, *Music and memory: An introduction*. MIT press, 2000 (cit. on pp. 22, 23, 64).

[100] S. L. Tanimoto, "Viva: A visual language for image processing," *Journal of Visual Languages & Computing*, vol. 1, no. 2, pp. 127–139, 1990 (cit. on pp. 22, 59).

[101] C. Nash and A. F. Blackwell, "Liveness and flow in notation use.," in *NIME*, 2012 (cit. on pp. 22, 59, 84).

[102] L. Church, C. Nash and A. F. Blackwell, "Liveness in notation use: From music to programming.," in *PPIG*, Citeseer, 2010, p. 2 (cit. on p. 22).

[103] P. Oliveros, S. Weaver, M. Dresser, J. Pitcher, J. Braasch and C. Chafe, "Telematic music: Six perspectives," *Leonardo Music Journal*, vol. 19, no. 1, pp. 95–96, 2009 (cit. on p. 23).

[104] C. Roberts and G. Wakefield, "Tensions and techniques in live coding performance.," 2018 (cit. on pp. 23, 30, 60, 73, 75, 83, 92, 101).

[105] T. Magnusson, "The ixi lang: A supercollider parasite for live coding," in *ICMC*, 2011 (cit. on p. 23).

[106] G. Dyson, *Analogia: the emergence of technology beyond programmable control*. Farrar, Straus and Giroux, 2020 (cit. on p. 23).

[107] A. F. Blackwell, "Bottom-up design and this thing called an 'object'," *EXE Magazine*, vol. 8, no. 7, pp. 28–32, 1993. [Online]. Available: `https://www.cl.cam.ac.uk/~afb21/publications/EXE93.html` (cit. on pp. 24, 25).

[108] I. m. zmölnig, "Audience perception of code," *International Journal of Performance Arts and Digital Media*, vol. 12, no. 2, pp. 207–212, 2016 (cit. on p. 24).

[109] P. Gross and C. Kelleher, "Non-programmers identifying functionality in unfamiliar code: Strategies and barriers," *Journal of Visual Languages & Computing*, vol. 21, no. 5, pp. 263–276, 2010 (cit. on p. 24).

[110] A. F. Blackwell, G. Cox and S. Lee, "Live writing the live coding book," in *Proc. First International Conference on Live Coding*, McMaster University, 2016 (cit. on pp. 24–26).

[111] F. Bernardo, C. Kiefer and T. Magnusson, "An audioworklet-based signal engine for a live coding language ecosystem," in *Web Audio Conference (WAC 2019)*, 2019, pp. 77–82 (cit. on p. 24).

[112] D. Griffiths, "Game pad live coding performance," *Die Welt als virtuelles Environment. Dresden: TMA Helleraue*, 2007 (cit. on pp. 24, 26, 43).

[113] J. Reus, *Imac music*, 2011. [Online]. Available: `https://jonathanreus.com/portfolio/imac-music/` (visited on 30/01/2022) (cit. on pp. 25, 43, 44).

[114] T. Bovermann and D. Griffiths, "Computation as material in live coding," *Computer music journal*, vol. 38, no. 1, pp. 40–53, 2014 (cit. on p. 26).

[115] A. Drymonitis, "Live coding poetry: The narrative of code in a hybrid musical/poetic context," *Organised Sound*, 1–12, 2023. DOI: `10.1017/S1355771823000493` (cit. on p. 26).

[116] S. W. Lee and G. Essl, "Live writing: Asynchronous playback of live coding and writing," in *Proceedings of the International Conference on Live Coding*, 2015 (cit. on pp. 26, 72).

[117] S. W. Lee, G. Essl and M. Martinez, "Live writing: Writing as a real-time audiovisual performance.," in *NIME*, 2016, pp. 212–217 (cit. on p. 26).

[118] N. Collins, "Live coding and machine listening," in *Proceedings of the First International Conference on Live Coding*, 2015, pp. 4–11 (cit. on p. 27).

[119] A. F. Blackwell, "Patterns of user experience in performance programming," in *Proc. First International Conference on Live Coding*, 2015 (cit. on pp. 28, 84).

[120] A. Veinberg and F. I. Noriega, *Coding With a Piano: the First Phase of the Codeklavier's Development*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2018 (cit. on p. 28).

[121] M. Mayas, *Orchestrating timbre–Unfolding processes of timbre and memory in improvisational piano performance*. 2019 (cit. on p. 29).

[122] M. Baalman, "Embodiment of code," in *Proceedings of the first international conference on live coding*, 2015, pp. 35–40 (cit. on pp. 30, 43, 44, 47).

[123] C. Palmer, "10 music performance: Movement and coordination," *The psychology of music (third edition)*, vol. 29, p. 405, 2012 (cit. on pp. 31, 81).

[124] R. B. Allen, "Mental models and user models," in *Handbook of human-computer interaction*, Elsevier, 1997, pp. 49–63 (cit. on p. 31).

[125] C. Kiefer, "Approximate programming: Coding through gesture and numerical processes," in *Proceedings of the First International Conference on Live Coding, ICSRiM, University of Leeds*, 2015 (cit. on pp. 33, 34, 43, 44, 74).

[126] T. Bovermann, J. Rohrhuber and A. de Campo, "Laboratory methods for experimental sonification," *The sonification handbook*, pp. 237–272, 2011 (cit. on p. 34).

[127] P. Dahlstedt, "Dynamic mapping strategies for expressive synthesis performance and improvisation," in *Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music: 5th International Symposium, CMMR 2008 Copenhagen, Denmark, May 19-23, 2008 Revised Papers 5*, Springer, 2009, pp. 227–242 (cit. on pp. 34, 44, 74).

[128] D. Wessel, "An enactive approach to computer music performance," *Le Feedback dans la Creation Musical, Lyon, France*, pp. 93–98, 2006 (cit. on pp. 34, 53).

[129] P. Dahlstedt, "Creating and exploring huge parameter spaces: Interactive evolution as a tool for sound generation," in *ICMC*, 2001 (cit. on p. 34).

[130] J. Armitage, A. McPherson *et al.*, "The stenophone: Live coding on a chorded keyboard with continuous control," 2017 (cit. on p. 38).

[131] A. R. Jensenius, M. M. Wanderley, R. I. Godøy and M. Leman, "Musical gestures: Concepts and methods in research," in *Musical gestures*, Routledge, 2010, pp. 24–47 (cit. on pp. 41, 47).

[132] S. Pinker, "How the mind works (1997/2009)," 2009 (cit. on p. 41).

[133] K. Sicchio, "Hacking choreography: Dance and live coding," *Computer Music Journal*, vol. 38, no. 1, pp. 31–39, 2014 (cit. on p. 43).

[134] S. Salazar and J. Armitage, "Re-engaging the body and gesture in musical live coding," 2018 (cit. on p. 43).

[135] M. A. Baalman, "Interplay between composition, instrument design and performance," *Musical Instruments in the 21st Century: Identities, Configurations, Practices*, pp. 225–241, 2017 (cit. on p. 43).

[136] F. I. Noriega and A. Veinberg, "The sound of lambda," in *Proceedings of the 7th acm sigplan international workshop on functional art, music, modeling, and design*, 2019, pp. 56–60 (cit. on pp. 43, 44).

[137] T. Magnusson, "Improvising with the threnoscope: Integrating code, hardware, gui, network, and graphic scores.," in *NIME*, 2014, pp. 19–22 (cit. on pp. 43, 45, 87).

[138] P. A. Nilsson, *A field of possibilities: Designing and playing digital musical instruments*. Academy of Music and Drama; Högskolan för scen och musik, 2011 (cit. on pp. 43, 52).

[139] H. Ishii and B. Ullmer, "Tangible bits: Towards seamless interfaces between people, bits and atoms," in *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, 1997, pp. 234–241 (cit. on p. 45).

[140] J. W. Davidson, "Visual perception of performance manner in the movements of solo musicians," *Psychology of music*, vol. 21, no. 2, pp. 103–113, 1993 (cit. on pp. 46, 48, 91).

[141] T. Sayer, "Cognition and improvisation: Some implications for live coding," 2015 (cit. on pp. 47, 86).

[142] A. Goldman, "Live coding helps to distinguish between embodied and propositional improvisation," *Journal of New Music Research*, vol. 48, no. 3, pp. 281–293, 2019 (cit. on pp. 47, 86).

[143] S. Salazar, "Searching for gesture and embodiment in live coding," in *Proceedings of the international conference on live coding*, 2017 (cit. on p. 47).

[144] C. C. Hutchins, "Live patch/live code," in *Proceedings of the first international conference on live coding*, 2015, pp. 147–151 (cit. on p. 47).

[145] R. I. Godøy, "Gestural-sonorous objects: Embodied extensions of schaeffers's conceptual apparatus," *Organised sound*, vol. 11, no. 2, pp. 149–157, 2006 (cit. on p. 47).

[146] D. Kahneman, *Thinking, fast and slow*. macmillan, 2011 (cit. on p. 48).

[147] S. L. Tanimoto, "Challenges for livecoding via acoustic pianos," in *3rd International Conference on Live Coding. Morelia, Mexico*, 2017 (cit. on pp. 49, 84).

[148] J. Malloch, D. Birnbaum, E. Sinyor and M. M. Wanderley, "A new conceptual framework for digital musical instruments," in *Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06)*, 2006, pp. 49–52 (cit. on p. 50).

[149] N. Rasamimanana, "Towards a conceptual framework for exploring and modelling expressive musical gestures," *Journal of New Music Research*, vol. 41, no. 1, pp. 3–12, 2012 (cit. on p. 50).

[150] T. Sayer, "Cognitive load and live coding: A comparison with improvisation using traditional instruments," *International Journal of Performance Arts and Digital Media*, vol. 12, no. 2, pp. 129–138, 2016 (cit. on pp. 54, 86).

[151] M. Kirkegaard, M. Bredholt, C. Frisson and M. Wanderley, "Torquetuner: A self contained module for designing rotary haptic force feedback for digital musical instruments," in *Proceedings of the international conference on new interfaces for musical expression*, 2020, pp. 273–278 (cit. on pp. 55, 102).

[152] A. McLean and G. Wiggins, "Computer programming in the creative arts," *Computers and Creativity*, pp. 235–252, 2012 (cit. on p. 57).

[153] P. Dahlstedt, "Between material and ideas: A process-based spatial model of artistic creativity," in *Computers and Creativity*, Springer, 2012, pp. 205–233 (cit. on p. 57).

[154] G. E. Lewis, "Too many notes: Computers, complexity and culture in voyager," *Leonardo Music Journal*, vol. 10, pp. 33–39, 2000 (cit. on p. 57).

[155] G. Assayag and S. Dubnov, "Using factor oracles for machine improvisation," *Soft Computing*, vol. 8, no. 9, pp. 604–610, 2004 (cit. on pp. 57, 58).

[156] P. Dahlstedt and P. McBurney, "Musical agents: Toward computer-aided music composition using autonomous software agents," *Leonardo*, vol. 39, no. 5, pp. 469–470, 2006 (cit. on pp. 57, 60).

[157] M. Minsky, "Music, mind, and meaning," *Computer Music Journal*, 1981 (cit. on p. 57).

[158] F. Castiglione, "Agent based modeling," *Scholarpedia*, vol. 1, no. 10, p. 1562, 2006, revision 123888. DOI: 10.4249/scholarpedia.1562 (cit. on p. 57).

[159] R. Rowe, "Machine listening and composing with cypher," *Computer Music Journal*, vol. 16, no. 1, pp. 43–63, 1992 (cit. on p. 57).

[160] G. Assayag, G. Bloch, M. Chemillier, A. Cont and S. Dubnov, "Omax brothers: A dynamic yopology of agents for improvization learning," in *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, 2006, pp. 125–132 (cit. on p. 58).

[161] F. Pachet, "The continuator: Musical interaction with style," *Journal of New Music Research*, vol. 32, no. 3, pp. 333–341, 2003 (cit. on p. 58).

[162] T. Gifford, S. Knotts, J. McCormack, S. Kalonaris, M. Yee-King and M. d'Inverno, "Computational systems for music improvisation," *Digital Creativity*, vol. 29, no. 1, pp. 19–36, 2018 (cit. on p. 58).

[163] K. Tatar and P. Pasquier, "Musical agents: A typology and state of the art towards musical metacreation," *Journal of New Music Research*, vol. 48, no. 1, pp. 56–105, 2019 (cit. on p. 58).

[164] A. Xambó, "Virtual agents in live coding: A short review," *arXiv preprint arXiv:2106.14835*, 2021 (cit. on pp. 58, 61, 89).

[165] C. Nash, "Supporting virtuosity and flow in computer music," Ph.D. dissertation, University of Cambridge, 2012 (cit. on p. 59).

[166] J. Croft, "Theses on liveness," *Organised Sound*, vol. 12, no. 1, pp. 59–66, 2007 (cit. on p. 59).

[167] P. Auslander, "Digital liveness: A historico-philosophical perspective," *PAJ: A journal of performance and art*, vol. 34, no. 3, pp. 3–11, 2012 (cit. on p. 59).

[168] S. J. Norman, *Senses of liveness for digital times*, 2016 (cit. on p. 59).

[169] A. C. Sorensen and A. R. Brown, "Aa-cell in practice: An approach to musical live coding," in *International Computer Music Conference*, 2007, pp. 292–299 (cit. on pp. 60, 62).

[170] A. Xambó, G. Roma, P. Shah, J. Freeman and B. Magerko, "Computational challenges of co-creation in collaborative music live coding: An outline," in *Proceedings of the 2017 Co-Creation Workshop at the International Conference on Computational Creativity*, 2017 (cit. on p. 61).

[171] J. Stewart and S. Lawson, "Cibo: An autonomous tidalcyles performer," in *Proceedings of the Fourth International Conference on Live Coding*, 2019, p. 353 (cit. on p. 61).

[172] S. Knotts, "Algorithmic interfaces for collaborative improvisation," in *Proceedings of the International Conference on Live Interfaces*, 2016, pp. 232–237 (cit. on p. 61).

[173] L. Navarro and D. Ogborn, "Cacharpo: Co-performing cumbia sonidera with deep abstractions," in *Proceedings of the International Conference on Live Coding*, 2017 (cit. on p. 61).

[174] N. Collins, "Scmir: A supercollider music information retrieval library," in *ICMC*, 2011 (cit. on p. 61).

[175] N. Collins and S. Knotts, "A javascript musical machine listening library," in *International Computer Music Conference (ICMC 2019)*, 2019 (cit. on p. 61).

[176] F. Bernardo, C. Kiefer and T. Magnusson, "A signal engine for a live coding language ecosystem," *Journal of the Audio Engineering Society*, vol. 68, no. 10, pp. 756–766, 2020 (cit. on p. 61).

[177] P. A. Tremblay, G. Roma and O. Green, "The fluid corpus manipulation toolkit: Enabling programmatic data mining as musicking," *Computer Music Journal*, 2022 (cit. on p. 61).

[178] A. Caillon and P. Esling, "Rave: A variational autoencoder for fast and high-quality neural audio synthesis," *arXiv preprint arXiv:2111.05011*, 2021 (cit. on p. 61).

[179] J. Shimizu, R. Fiebrink *et al.*, "Genny: Designing and exploring a live coding interface for generative models," 2023 (cit. on p. 61).

[180] F. A. Dal Rì and R. Masu, "Exploring musical form: Digital scores to support live coding practice," in *NIME 2022*, PubPub, 2022 (cit. on pp. 64, 82, 87).

[181] T. Magnusson, "Code scores in live coding practice," in *Proceedings of the International Conference for Technologies for Music Notation and Representation, Paris*, vol. 5, 2015 (cit. on pp. 64, 82).

[182] *Strudel: Live Coding Patterns on the Web*, Alex McLean's work on this project is supported by a UKRI Future Leaders Fellowship [grant number MR/V025260/1]., Zenodo, Apr. 2023. DOI: 10.5281/zenodo.7842142. [Online]. Available: https://doi.org/10.5281/zenodo.7842142 (cit. on p. 65).

[183] C. Roberts and J. Kuchera-Morin, "Gibber: Live coding audio in the browser," in *ICMC*, vol. 11, 2012, p. 6 (cit. on p. 65).

[184] A. McLean, D. Griffiths, N. Collins and G. Wiggins, "Visualisation of live code," *Electronic Visualisation and the Arts (EVA 2010)*, pp. 26–30, 2010 (cit. on pp. 65, 87).

[185] S. Knotts *et al.*, "Social systems for improvisation in live computer music," Ph.D. dissertation, Durham University, 2018 (cit. on p. 65).

[186] S. O. Hansson, "Risk," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta and U. Nodelman, Eds., Winter 2022, Metaphysics Research Lab, Stanford University, 2022 (cit. on pp. 71, 73).

[187] C. Nash, "The cognitive dimensions of music notations," 2015 (cit. on pp. 72, 74).

[188] A. F. Blackwell and N. Collins, "The programming language as a musical instrument.," in *PPIG*, 2005, p. 11 (cit. on p. 72).

[189] A. Ward, J. Rohrhuber, F. Olofsson *et al.*, "Live algorithm programming and a temporary organisation for its promotion," in *Proceedings of the README Software Art Conference*, vol. 289, 2004, p. 290 (cit. on pp. 72, 73).

[190] J. Reus and J. Chicau, "Anatomical intelligence: Live coding as performative dissection," *Organised Sound (28) 2 (In Press)*, 2023 (cit. on p. 72).

[191] A. McLean and G. A. Wiggins, "Texture: Visual notation for live coding of pattern," in *ICMC*, 2011 (cit. on p. 72).

[192] S. Aaron, A. F. Blackwell, R. Hoadley and T. Regan, "A principled approach to developing new languages for live coding.," in *NIME*, 2011, pp. 381–386 (cit. on p. 74).

[193] T. Magnusson and K. Sicchio, *Writing with shaky hands*, 2016 (cit. on p. 75).

[194] A. McLean, "Stress and cognitive load," in *Collaboration and learning through live coding*, A. Blackwell, A. McLean, J. Noble and J. Rohrhuber, Eds., 2014, pp. 145–146 (cit. on p. 75).

[195] J. Chicau and J. Reus, "Anatomical intelligence: Live coding as performative dissection," *Organised Sound*, 1–15, 2023. DOI: 10.1017/S1355771823000481 (cit. on p. 76).

[196] A. F. Blackwell and S. Aaron, "Craft practices of live coding language design," in *Proc. first international conference on live coding*, Zenodo, 2015 (cit. on p. 78).

[197] S. Gibet, "Sensorimotor control of sound-producing gestures," in *Musical gestures*, Routledge, 2010, pp. 224–249 (cit. on pp. 81, 82).

[198] M. Lesaffre, E. Van Dyck and M. Leman, *Expressive interaction with music*, 2019 (cit. on p. 81).

[199] K. Burland and A. McLean, "Understanding live coding events," *International Journal of Performance Arts and Digital Media*, vol. 12, no. 2, pp. 139–151, 2016 (cit. on p. 82).

[200] R. Bell, "Towards useful aesthetic evaluations of live coding," in *ICMC*, 2013 (cit. on p. 82).

[201] E. Wilson, G. Fazekas and G. Wiggins, "On the integration of machine agents into live coding," *Organised Sound*, 1–10, 2023. DOI: 10.1017/S1355771823000420 (cit. on p. 82).

[202] A. Cárdenas, "Street code-live coding in public space," in *Proceedings of the International Conference on Live Coding*, ICLC, 2019 (cit. on p. 82).

[203] H. Villaseñor-Ramírez, "Live coding outside, live coding inside: Listening, participation and walking," *Organised Sound*, 1–11, 2023. DOI: 10.1017/S1355771823000353 (cit. on p. 82).

[204] C. Schuster and C. Flanagan, "Live programming by example: Using direct manipulation for live program synthesis," in *LIVE Workshop*, 2016 (cit. on pp. 84, 91).

[205] C. Roberts, "Realtime annotations & visualizations in live coding performance," in *Proceedings of the 2018 LIVE Programming Workshop*, 2018 (cit. on p. 87).

[206] M. Sturdee and J. Lindley, "Sketching & drawing as future inquiry in hci," in *Proceedings of the Halfway to the Future Symposium 2019*, 2019, pp. 1–10 (cit. on p. 87).

[207] U. Attanayake, B. Swift, H. Gardner and A. Sorensen, "Disruption and creativity in live coding," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2020, pp. 1–5 (cit. on p. 92).

[208] E Wilson, S Lawson, A McLean, J Stewart *et al.*, "Autonomous creation of musical pattern from types and models in live coding," 2021 (cit. on p. 92).

[209] Z. Rakhimberdina, Q. Jodelet, X. Liu and T. Murata, "Natural image reconstruction from fmri using deep learning: A survey," *Frontiers in neuroscience*, vol. 15, p. 795 488, 2021 (cit. on p. 92).

[210] L. Bellier, A. Llorens, D. Marciano *et al.*, "Music can be reconstructed from human auditory cortex activity using nonlinear decoding models," *PLoS biology*, vol. 21, no. 8, e3002176, 2023 (cit. on p. 92).

[211] R. Bell, "A live coding improvisation," in *Proceedings of the 9th ACM Conference on Creativity & Cognition*, 2013, pp. 392–393 (cit. on p. 95).

[212] S. W. Lee, J. Freeman, A. Colella, S. Yao and A. Van Troyer, "Collaborative musical improvisation in a laptop ensemble with lolc," in *Proceedings of the 8th ACM Conference on Creativity and Cognition*, 2011, pp. 361–362 (cit. on p. 95).

[213] B. Swift, A. Sorensen, M. Martin and H. Gardner, "Coding livecoding," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2014, pp. 1021–1024 (cit. on p. 95).

[214] J. Françoise, S. Fdili Alaoui and Y. Candau, "Co/da: Live-coding movement-sound interactions for dance improvisation," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–13 (cit. on p. 95).

[215] C. Frisson and M. M. Wanderley, "Challenges and opportunities of force feedback in music," in *Arts*, MDPI, vol. 12, 2023, p. 147 (cit. on p. 102).

[216] C. Erdem, B. Wallace and A. R. Jensenius, "Cavi: A coadaptive audiovisual instrument–composition," in *NIME 2022*, PubPub, 2022 (cit. on p. 102).

[217] P. T. Daniels and W. Bright, *The world's writing systems*. Oxford University Press, 1996 (cit. on p. 102).

Appended Papers

Paper I

An analytical framework for musical live coding systems based on gestural interactions in performance practices

G. Diapoulis and P. Dahlstedt

Proceedings of the International Conference on Live Coding (ICLC).
Valdivia, Chile. (2021).

https://doi.org/10.5281/zenodo.5801942

Proceedings of the International Conference on Live Coding, Valdivia, December 2021
https://iclc.toplap.org/2021/

An analytical framework for musical live coding systems based on gestural interactions in performance practices

Georgios Diapoulis
Interaction Design, Department of Computer Science & Engineering, Chalmers University of Technology, University of Gothenburg
geodia@chalmers.se

Palle Dahlstedt
Interaction Design, Department of Computer Science & Engineering, Chalmers University of Technology, University of Gothenburg
palle@chalmers.se

ABSTRACT

Gestural interaction in live coding performance is still in its infancy, albeit the long tradition in music performance studies. Computational challenges in musical live coding have been motivating the research community towards the development of novel programming languages and interfaces. On the other hand, given the maturity of many music systems there is an increasing demand for theory building on live coding systems and practices. Here, we present an observational study from videos of live performances available online and we introduce an analytical framework for live coding music systems. We begin by examining how performance practices differ on potentially equivalent systems. On the spotlight of the framework is the viewpoint of gestural interactions under the prism of music psychology and perception. We examined several systems on three main processes: (i) interface design, (ii) gestural mapping and (iii) user's interaction. These processes are presented as an orthogonal three-dimensional framework, so to facilitate visualizations and readers' understanding. Preliminary assessments of the systems in question agree with ground truth knowledge of the computational classification of the systems. Furthermore, we analyze a few notable systems that are stretching the boundaries of our dimensional framework, indicating that more dimensions may be required. Finally, we discuss the analytical framework in relation to a higher-level description of live coding music performance and we discuss future studies that may be conducted to assess the validity of this approach.

1. INTRODUCTION

Gestures are an integral part in our daily interactions with computing technologies. There has been increasing interest in mobile devices that afford gestural interactions, typically through touchscreen displays, which in return are progressively transforming user's interactions. In psychology of programming, users are seen as experts (Blackwell & Collins, 2005) and users' practices may influence technological developments. In analogy to this liberated stance of the division between users versus experts, here, we take a liberated stance on live coding. We adhere any declarations of live coding practices as live coding, without a need for strict definitions (Collins, 2011). In the context of music performance, gestures have been studied extensively in both music psychology and music perception. There is a broad consensus that musical gestures carry functional, aesthetical and social aspects (Godøy & Leman, 2010).

1.1 Gestural interaction in live coding research

In live coding, gestural interactions are seen as an indispensable part which can be improved in terms of virtuosity and expressivity by extensive practice (Collins, 2011). Yet, since live coding as performance practice is still in its infancy, there are no methods on how to master gestural control in performance. Many practitioners have reported embodiment during live coding performance. More specifically, Baalman (2015) reflects on typing automaticity that is developed through the familiarity

with a certain programming language. Given such anecdotal evidence, action execution is linked to the mental model that we form by extensive practice with the programming language of our preference.

In the context of traditional music performance, action execution is linked to imagery of auditory percepts (Keller & Koch, 2008). Such, imagery percepts are fed to gestural unfoldings as these are realized by sequences of musical gestures during performance. Auditory and motor perception are the driving forces of these realizations and both effortful and involuntary imagery may be contributing factors during music making in live coding (Diapoulis & Dahlstedt, 2021). Given the extensive study and experimental evidence on auditory and motoric skills during performance, musical interface design studies have been taking advantage and building numerous evaluation frameworks for music systems. Gestural control in computer music interfaces has already a long tradition and there are numerous frameworks and evaluation studies. Indicatively, in the NIME community[1] a meta-analysis revealed that within the years 2012-2014 more that 200 studies related to evaluation had been reported (Barbosa et al., 2015).

Evaluation systems in musical interaction design have been proposed since the 90s, and in 00s there is already a steady ground which has a parallel coevolution with trends in human-computer interaction (Wanderley & Orio, 2002). Early studies on evaluation of musical interfaces have been focusing on assessments of simple interactions. Later, developments of mixed methods were employed to evaluate authenticity of artificial agents (Stowell et al., 2009). Despite all these efforts, it has been noted that most systems are used by a single performer, who is typically the developer of the system (Barbosa et al., 2015). This makes difficult to build a solid background on evaluation methods of digital musical instruments (DMIs). In the same study were identified some basic components among the reviewed articles, which are related to the *criteria* of the evaluation, the *methods* used and the *goals* of the evaluations among others.

2. CRITERIA, GOALS AND METHODS FOR AN ANALYTICAL FRAMEWORK

Here, we will present the criteria, goals and methods of our study. The main *goal* is to present a preliminary analytical framework for live coding music systems. Our starting point are the gestural interactions with the musical interface. The *method* is based on an observation study of videos available online (see Table 1). The main *criteria* are (i) to identify performance practices that show broad variations on the gestural interactions, and (ii) to examine how performance practices may differ in potentially equivalent systems. In that manner, we did not include a broad variety of "standard" or "canonical" (Roberts and Wakefield, 2018) live coding systems, because they demonstrate minimal variation in gestural interactions. By "standard live coding" systems, we address all practices that are using the keyboard as the main input interface and a typical programming language. Here, a typical programming language, refers to any programming language which can make use of an interpreter or a compiler, and requires typed programming expressions that are well-formed in a text editor. Instead, in this study we focused on highly individualistic systems that build upon anything from low-level computing interfaces to high-level systems. These may include from traditional musical instruments, like the piano, to printed circuit boards and hardware prototypes on solderless breadboards.

	Author	System	Video
1	Baalman	Code LiveCode Live	https://vimeo.com/434679284
2	Collins	Type-A personality	https://youtu.be/0fX0AymCtgA
3	Diapoulis	stateLogic machine	https://vimeo.com/43121821

[1]New Interfaces for Musical Expression, https://www.nime.org/.

4	Griffiths	Al-jazzari	https://youtu.be/Uve4qStSJq4
5	Magnusson	Threnoscope	https://vimeo.com/63335988
6	McLean	TidalCycles	https://youtu.be/PeyE8ATMezs
7	Noriega & Veinberg	CodeKlavier CKalkulator	https://youtu.be/hD-PWNDebD4
8	Noriega & Veinberg	CodeKlavier hello world	https://youtu.be/ytpB8FB6VTU
9	Reus	iMac Music	https://vimeo.com/205714278
10	Salazar	Auraglyph	https://youtu.be/qqt2vSNy_nA

Table 1. Video material for the observational study.

2.1 Criteria

The key criterion for the analytical framework is the gestural interaction with the musical interface. More specifically, we aim to identify whether the gestural interactions have any impact on the running algorithm of the system. Given the anecdotal evidence that a programming language may influence action (motor) execution (Baalman, 2015), it is reasonable to think how execution of gestural interactions can change our mental model of the running algorithm. Also, we have identified that there is increasing interest in cases where performance practices show variations on potentially equivalent systems (Diapoulis & Dahlstedt, 2021). Furthermore, to ease the theoretical analysis we excluded any visual percepts and we focused only on auditory and motor perception.

2.2 Goals

Our goal is to present a preliminary analytical framework for musical live coding systems. We coupled this framework to a theoretical background which may account for a high-level description of live coding music performance. Furthermore, we aim to systematize the study of gestural interactions in live coding performance, which may support the development of future experimental studies in the psychology and perception of live coding.

2.3 Methods

The main method was that we coded videos of live coding performances, based on subjective evaluations of the first author (see Table 3). Having in mind the key criterion of gestural interactions, we attempt to identify which might be the most important factors that differentiate these performance systems. Also, we constrained these factors up to three to facilitate visual communication and reader's understanding. Our method differs from previous studies in the live coding community. These have been ranging from aesthetic evaluation studies (Bell, 2013), to position articles on interface design (Stowell & McLean, 2013) and theoretical approaches on musical gestures (Jarvis, 2019) and cognitive processes (Sayer, 2015). Here, we present an alternative view based on analyzing videos online and we attempt to bridge studies from music psychology and perception within live coding research (Tanimoto, 2017). Our view stems from embodied music cognition (Leman, 2008) which moves beyond a view of input-output processes of human perception and cognition, and on previous work on embodied playing with algorithms (Dahlstedt, 2018). On this background and given a subjective perspective of the first author on live coding practices we are presenting a preliminary analytical framework of musical live coding systems. From the observed video material that is available online (see Table 1) we extracted meaningful abstractions and delivered a preliminary framework that can be discussed, challenged modified and extended by the live coding community. While we also discuss on live coding practices and agents the focus on this study is on live coding systems.

3. SETTING THE GROUND FOR AN ANALYTICAL FRAMEWORK

In this section, we start with a high-level description of live coding music performance. Then, we present our view on how music psychology and music perception may be related to live coding practice. Following that, we try to link this view to music cognition and agency in live coding. Finally, in the next section we introduce a preliminary dimensional framework for musical live coding systems.

We see that a three-fold description of practices, agents and systems is at the very center of studies in live coding music performance (see Figure 1, bottom row). The human companion is an indispensable part in live coding practice (Collins et al., 2003). This is because even if the musical agents may be fully autonomous still the training corpus is based on code written by humans (Stewart and Lawson, 2019). Although we may imagine fully-autonomous systems which may not be based on humanly-written code for the training data, yet, some amount of human agency is being transferred from the very foundations of computer science (Dahlstedt, 2021). Also, aspects of human-machine musicianship that arise during a live coding performance are shifting the boundaries of agency in music performance (Brown, 2016), and sophisticated designs may offer symbionts of human-machine musicianship (Collins, 2015). Finally, a music system seems to be a necessary component for musical live coding performances, regardless of seeing this as a musical notation system or a formal computational language (Magnusson, 2011).
Music systems show a broad diversity from low level computing components to high level languages and interface setups. Consequently, practices also show a broad diversity which is influenced to some extent by the design decisions of the systems. Finally, agents within the context of live coding music performance also show a broad diversity, ranging from human agents, either expert/novice programmers or expert/novice musicians, to artificial autonomous agents based on machine learning and machine listening. Indicatively, Nick Collins presented two alternatives, which may also overlap to some extent, as machine listening control of live coding, or live coding control of machine listening (Collins, 2015).

3.1 Musical activities in live coding practice

We start our high-level description of live coding performance by discussing how musical activities may be experienced in live coding practice. Musical activities can be divided to three overlapping categories, music making, music listening and musical imagery (Luck, 2015). We may also claim that there is a progressive level of engagement within musical activities, starting from least engagement in musical imagery, more engagement into music listening and even more in music making (see Figure 1, Activity). Music making involves both music composition and music performance. Skilled musical performance is achieved when the musician is exposed to repetitive practice of an activity. The same applies to live coding practice (Nilson, 2007).

3.2 Music perception in live coding performance

3.2.1 Music listening and appreciation

Music listening is the most wide-spread musical activity. We are exposed many hours per day to music, even involuntary. During music performance, either traditional or computer music performance, the musician is both making and listening to the generated sounds and appreciates online music percepts. Contrary to traditional music performance, in computer music the generation of sounds is cloaked within circuits and other high-level components. Thus, in the absence of any visual and haptic feedback (which is typically predominant in traditional music performance), in computer music we are exclusively relying in our auditory capabilities. This is done by taking advantage of our ability to segment sound events with audition. While onset and offset detection can be a notoriously difficult task for machine listening, and may be a philosophical enquiry on its own (Toiviainen, 2015; personal

communication), in computer generated music there is no alternative, other than to embody sound events just by listening to them (Palmer, 1997). As a result, here, we excluded any visual percepts during musical live coding. This is because the visual channel is a another complicated mechanism and during live coding it can have a very important contribution to the experience. This decision to focus on motor and auditory percepts may also contribute to the discussion of live coding for blind and visually impaired people (Vetter, 2020).

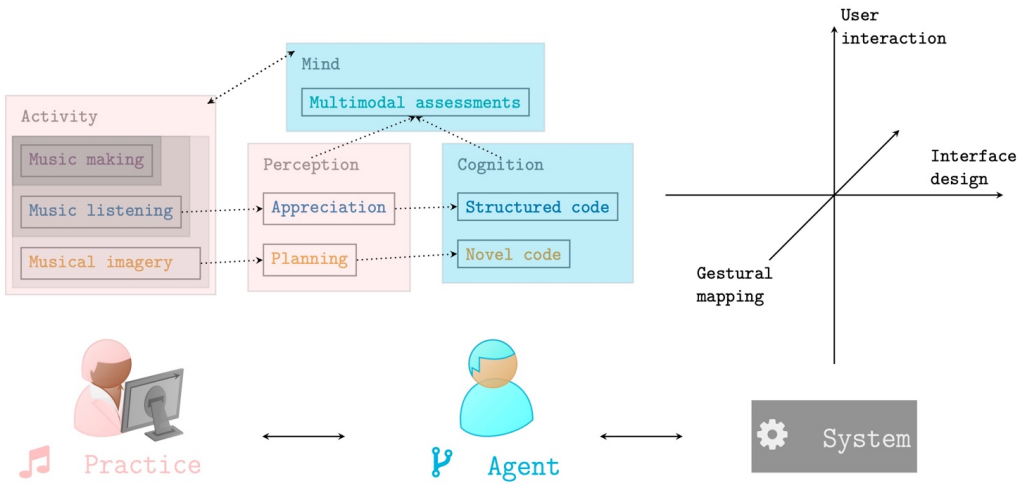


Figure 1. High-level description of live coding music performance: practices, agents, systems; including the subcomponents of musical activities, music perception and cognition in performance and the dimensional analytical framework of live coding systems.

3.2.2 Musical imagery and planning

Musical imagery is when a melody of a song comes to our mind. The so-called earworms, exemplify the phenomenon of involuntary musical imagery. During music performance, we employ both involuntary and effortful music percepts (Keller, 2012). Interestingly, involuntary imagery can also trigger motor execution (Keller & Koch, 2008). Thus, a blend of effortful and involuntary imageries take place during performance. Musical imagery is intertwined to anticipated music percepts and Godøy (2003) has suggested that gestural imagery is an integral part in music performance. He elaborates on that and hints that our mental capacity enable us to compress gestural unfoldings in time, while the same is not true for auditory percepts. One cannot compress a musical sound and experience the same qualities. In live coding we are planning our future actions by trial-and-error, which is an act of novelty (Tanimoto, 2017). On this account, auditory percepts can trigger gestural unfoldings during a live coding performance, which in return may contribute to planning of abstract actions (Diapoulis & Dahlstedt, 2021). Interestingly, musical notation can also trigger involuntary imagery, which is known as notational audiation in music literature (Keller, 2012).

3.3 Music cognition and code structures

As presented above, musical planning is a combination of gestural and auditory anticipatory percepts and music listening can be employed to experience segmentations of the generated sounds. Also, when we appreciate a music percept, this may result to structured code in live coding practice (see Figure 1,

Cognition). The question arises how such sounds may be used to change our mental model of the running program? Here, sound segmentation is seen as the only informative unit which we can employ to modify and structure our code. In fact, algorithms are not structures which afford segmentation themselves. Furthermore, gestures also do not exhibit 'well formed' characteristics and it can be difficult even for human annotators to segment gestural unfoldings. Here, we see that segmentations that are formed from auditory percepts can inform gestural interactions in planning future action executions. In that manner, planning contributes to novel code evaluations (Diapoulis & Dahlstedt, 2021).

3.4 How agency appears in live coding

Artificial agency is an immediate consequence during live coding practice (Brown, 2016). This is further entangled when machine learning or machine listening components are involved during the activity of music making (Collins, 2015). For instance, when a machine listening component is being involved, then we can think of this as an augmentation to our hearing. When we control code structures or parameters with machine listening processes then the running program can be thought as an agent that applies semantic (code) adjustments that are driven by another modality. In that manner, the electronics are producing mechanical energy which vibrates air molecules and surrounding structures and then is processed again through digital logic to apply adjustments on its own structure. In fact, this can be also done without rendering the physical sounds, but we exemplify the physical process to facilitate reader's understanding. These practices are shifting conventional agencies in musicianship. In traditional music performance the musicians can embody expressive intentions in a clear manner and the visual channel biases our perception of expressivity (Davidson, 1993). How may we study expressivity in such symbionts of human-machine musicianship?

4. LIVE CODING SYSTEMS

A live coding system is a rather complicated structure. All the fruitful efforts of the live coding community to deliver systems that can enable live performances and even algorave parties has come to a rather matured state, in comparison to 10 years ago. Some of the most prominent systems are, for example, ixi-lang (Magnusson, 2011), SonicPi (Aaron, 2016) and TidalCycles (McLean & Wiggins, 2010). A common feature of all abovementioned systems is that they are based on elegant code expressions which foster immediacy during performance and may also ease educational purposes. Indicatively, ixi-lang was developed with a constraint of 5 seconds per command so to facilitate live performance. The technological demands of such developments and the fact that the original authors had to develop most of the system on their own, may have hindered other aspects of these developments. For instance, from the viewpoint of gestural interactivity these systems demonstrate equivalence at first glance. This is because, the composer-programmer is typing programming commands on a keyboard, which includes algorithmic complexity, and waiting until she has an executable command ready to be successfully evaluated by the interpreter. Here, we refer to this category of systems as “standard live coding” systems.

4.1 A three-dimensional analytical framework for musical live coding systems

In this section, we present a three-dimensional analytical framework (Diapoulis & Dahlstedt, 2021), equipped with a dyadic condition, here called code-first and music-first, as proposed by Tanimoto (2017). Each dimension represents a process as shown in Table 2, and is equipped with two semantic differential concepts. On the lower end is a low-level concept, also called concrete, and on the upper end a high-level concept, also called abstract. For the dyadic condition we assigned code-first as a low-level concept in analogy to how a musical score affords different interpretations during music performance. Algorithms are seen as scores in live coding (Magnusson, 2011), and in this case code is

seen as more concrete in comparison to the generated music which may differ for example in different music halls, sound reproduction systems and the like.

	Process	Low-level (concrete)	High-level (abstract)
X axis	Interface design (ID)	Literal design	Metaphorical design
Y axis	Gestural mapping (GM)	Algorithmic significance	Algorithm agnostic
Z axis	User interaction (UI)	Direct manipulation	Algorithmic complexity
Dyadic condition	Sound generation (SG)	Code-first	Music-first

Table 2. Dimensional analytical framework for musical live coding systems.

The first dimension, interface design, refers to the concept of how literal or metaphorical is the design of the interface. By literal design we refer to any design decisions that rely on conventional programming interfaces, like a text editor or a hardware prototype equipped with printed circuit boards, switches, buttons and the like. Metaphorical design refers to any design decisions which may conceal the programming activity, like playing the piano or playing a video game. The second dimension on gestural mapping examines what is the effect of gestural interactions on the running algorithm. For instance, during a "standard live coding" the musician is typing on the keyboard without any temporal or other constrains. In that manner, the live coder is doing as many gestures as she likes until the code execution is successful. We call this process algorithm agnostic. On the other hand, if the gestural unfoldings are modifying the structure of the running algorithm we call this process algorithmic significant. The best example to understand this dimension is to watch the performances by Noriega & Veinberg "hello world" and "CKalkuλator". In these two different setups of the CodeKlavier system, the pianist is typing on the musical keyboard (see "hello world"), or just playing the piano (see "CKalkuλator"). We see that in the "hello world" performance the gestural mapping is agnostic to the algorithm, while in "CKalkuλator" the gestures are modifying the running algorithm. Finally, the third dimension shows the semantic differentials of direct manipulation and algorithmic complexity. Defining direct manipulation in the context of live coding can be a challenging endeavour. We see that a musical interface which facilitates recognition instead of retrieval, may be classified as exhibiting direct manipulation.

Table 3 below shows a color coding for the systems examined in this study. The uppercase "L" stands for low-level concepts and the uppercase "H" for high-level concepts. When the systems in question afford both low-level and high-level concepts, we coded such cases as "L/H".

	Author	System	ID	GM	UI	SG
1	Baalman	Code LiveCode Live	L	H	H	H
2	Collins	Type-A personality	L/H	L/H	L	H
3	Diapoulis	stateLogic machine	L	L	L	L
4	Griffiths	Al-jazzari	H	L	L	L
5	Magnusson	Threnoscope	L/H	H	L/H	L
6	McLean	TidalCycles	L	H	H	L
7	Noriega & Veinberg	CodeKlavier CKalkuλator	H	L	H	H
8	Noriega & Veinberg	CodeKlavier hello world	H	H	H	H
9	Reus	iMac Music	L	L	L	H

10	Salazar	Auraglyph	H	H	H	L

Table 3. "L" for low-level concepts and "H" for high-level concepts. "ID": interface design, "GM": gestural mapping, "UI": user interaction, "SG": sound generation.

5. VISUAL REPRESENTATION OF LIVE CODING SYSTEMS IN A DIMENSIONAL FRAMEWORK

Figure 2 shows a three-dimensional representation of the analytical framework. This is based on an orthogonal coordinate system, which may be misleading for the reader as orthogonality typically refers to independent concepts. It is interesting to observe that in this spatial representation Marije's Baalman "Code LiveCode Live" system overlaps to a "standard live coding" system (see Table 3, McLean). Also, the "stateLogic machine" by Diapoulis overlaps with "iMac Music" by Reus. Finally, the performance "hello world" overlaps with "Auraglyph" system. Interestingly, in all these pairs the systems differ on the dyadic condition code-first and music-first. From the observation that the "Code LiveCode Live" differs from a "standard live coding" system only in the code-first condition, Marije's system assigned meaning to the act of typing on the keyboard (Diapoulis & Dahlstedt, 2021). What more can we learn from these overlapping systems that differ only on the dyadic condition code-first, music-first?

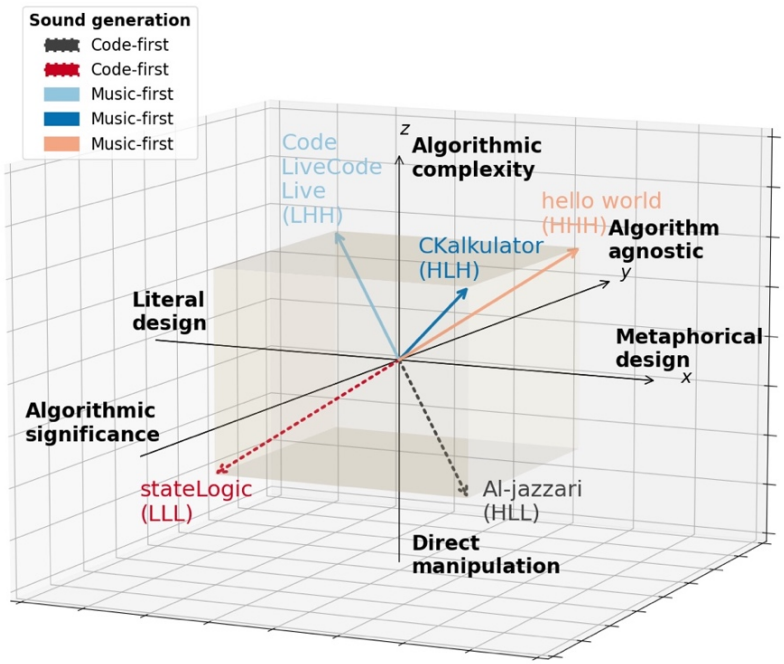


Figure 2. Three-dimensional analytical framework for musical live coding systems. X: interface design (ID), Y: gestural mapping (GM), Z: user interaction (UI). The dyadic condition on sound generation (SG) is shown with dashed arrows.

Based on Table 3 and Figure 2 we should note that a preliminary validation of the systems in question is provided by the directionality of the cognitive design that we selected for the semantic differentials. The classification as low-level and high-level on each system agrees to ground truth knowledge of the

workings of the systems. For instance, "Al-jazzari" is indeed based on low-level computing processes and is classified as such in all dimensions except the interface design (ID) axis.

6. DISCUSSION

We have introduced a preliminary three-dimensional analytical framework for musical live coding systems. Here, we attempt to bridge the gap between studies in music psychology and perception and to shift perceptions on fundamental differences between traditional and live coding music performance (Sayer, 2015; Tanimoto, 2017). The focal point of the study was to examine live coding systems from the viewpoint of gestural interactions. That was a revealing point of departure as we identified how potentially equivalent systems can bring about meaning to gestural interactions in live coding performance (Diapoulis & Dahlstedt, 2021).

We coded videos from performance practices available online, based on subjective evaluations of the first author. We identified early on, based on personal experience, that metaphorical design is particularly important within the broad diversity of live coding systems. Such systems can be quite surprising for someone somehow familiar with live coding, when she attends a live coding performance in which the performer seems to be engaged in a video game activity. Furthermore, based on personal reflections when designing a live coding prototype, it is reasonable to experience the dominance of the code-first requirement. When designing his "stateLogic machine", the first author (GD) tried to minimize this notably anxious waiting time, but he realized that you must always wait for the next positive edge clock. At least we cannot see any other way except if we move to a different computational paradigm that the input information is read on-demand and not periodically in time. About the second dimension on gestural mapping, we reflected on the literature of musical gestures in music performance (Godøy & Leman, 2010). In analogy to the primary and secondary aspects of musical gestures, we thought how musical gestures in live coding can be significant to the running algorithm. In traditional music performance primary gestures refer to sound-producing gestures, whereas the secondary gestures typically may carry emotions or facilitate communication. During live coding, the sound-producing part cannot correspond to the traditional meaning of sound-producing gestures, as the sound generation is performed using digital signal processing algorithms. Here, we identified that musical gestures in live coding can either have immediate impact on the running algorithm or can be ignorant about the workings of the algorithm. For instance, in Baalman's performance the musical gestures are sound-producing gestures, but they do not change the structure of the running algorithm. This because the modifications are performed on the parameter level. Contrary, in CKalkuλator the pianist is programming by playing the piano. Here, there is a fine line between what we adhere as programming or not. For example, in Threnoscope, Thor Magnusson is applying direct manipulation with the mouse to modified parameters of the running algorithm. We also classified this interaction as algorithm agnostic, due to the fact it does not apply changes to the structure of the running algorithm. A structural change on the running algorithm, is exemplified by Kiefer's (2015) "approximate programming". Specifically, this is exemplified by the real-time visualizations of the synths structure as these are shown using hierarchical trees[2]. An important note is that in computer music an algorithm is expressed using binary digits. One may question how a gesture may have an effect on a one-dimensional structure (Collins, 2016). Here, we see an algorithm as an abstraction that corresponds to a mental model and may be influence by embodiment in performance (Fanfani et al., 2020). Finally, the third dimension on user interaction was employed by studies in psychology of programming and human-computer interaction (Diapoulis & Dahlstedt, 2021). Sometimes, it can be difficult to evaluate whether a musical interface affords direct manipulation or algorithmic complexity. It should be noted that interfaces which support recognition processes are seen as exhibitors of direct manipulation.

[2] https://github.com/chriskiefer/ApproximateProgramming_SC/blob/master/approxTree.scd

On the dyadic condition code-first/music-first we would like to make an analogy to traditional musical instruments. Let us think for a moment how musical instruments like the qanun or the organ are changing music systems during performance. In both instruments the musician apply on-the-fly changes to the system. The qanun has a mandal technology which enables the performer to adjust the length of the strings. In principle, the mandal technology is changing the melodic modes, also known as maqams, that the instrument affords. Is this mandal technology a precursor to changing the program as it is running? Is this form of interaction a precursor of the dyadic condition code-first and music-first? Certainly, moving a mandal on the qanun is not a sound-producing gesture, but a necessary one.

In our study there is a predominant absence of collaborative live coding systems and practices. We did not examine this category of systems intentionally, due to the broad new perspective that collaborative live coding brings about in the development of the field. Performing with other people is seen as one of the most difficult tasks. If we would like to represent collaborative performance systems with the approach presented here, it is reasonable to assume that more dimensions are required for a more inclusive framework. A straight-forward solution to this can be a multi-dimensional space based on design space analysis (Birnbaum et al., 2005).

An important result from the present study is that a preliminary validation is provided by the computational classification of the systems, which agrees to ground truth knowledge. For instance, systems that are based on low-level computing processes are also classified as low-level systems in our dimensional framework. This is because we followed a cognitive paradigm which assigns the directionality of the dimensions from low-level to high-level concepts.

Finally, future studies should validate the framework and propose new dimensions for a more inclusive framework. Such frameworks may be used by both practitioners and academics for either creative explorations or theoretical discussion and design of experimental studies. This direction will contribute to the psychology and perception of live coding. For instance, interview studies with the original authors, or other live coders, may be employed to verify shared conceptions among the community. Also, questionnaire studies may validate, or not, the semantic differentials. If such efforts proved successful, then we may have to start thinking how we may browse such music spaces during live coding practices (ie. Type-A personality and Threnoscope). Creative exploration of such dimensional spaces should be seen as a helpful tool and as a challenge to move beyond the expressive capacity of such developments.

Acknowledgments

Special thanks to Mafalda Samuelsson-Gamboa for reviewing and Kivanç Tatar for input on this article.

REFERENCES

Aaron, Sam. "Sonic Pi–performance in education, technology and art." International Journal of Performance Arts and Digital Media 12, no. 2 (2016): 171-178.

Baalman, Marije. "Embodiment of code." In Proceedings of the First International Conference on Live Coding, pp. 35-40. 2015.

Barbosa, Jeronimo, Joseph Malloch, Marcelo M. Wanderley, and Stéphane Huot. "What does" Evaluation" mean for the NIME community?." (2015).

Bell, Renick. "Towards useful aesthetic evaluations of live coding." In ICMC. 2013.

Birnbaum, David, Rebecca Fiebrink, Joseph Malloch, and Marcelo M. Wanderley. "Towards a dimension space for musical devices." In Proceedings of the 2005 conference on New interfaces for musical expression, pp. 192-195. 2005.

Blackwell, Alan F., and Nick Collins. "The Programming Language as a Musical Instrument." In PPIG, p. 11. 2005.

Brown, Andrew R. "Performing with the other: the relationship of musician and machine in live coding." International Journal of Performance Arts and Digital Media 12, no. 2 (2016): 179-186.

Collins, Nicolas. "Semiconducting: Making music after the transistor." Musical Listening in the Age of Technological Reproduction. Routledge, 2016. 313-322.

Collins, Nick, et al. "Live coding in laptop performance." Organised sound 8.3 (2003): 321-330.

Collins, Nick. "Live coding of consequence." Leonardo 44, no. 3 (2011): 207-211.

Collins, Nick. "Live Coding and Machine Listening." In Proceedings of the International Conference on Live Coding, pp. 4-11. 2015.

Dahlstedt, Palle. "Action and Perception: Embodying Algorithms and the Extended Mind." The Oxford Handbook of Algorithmic Music, Oxford University Press, 2018. 41-65.

Dahlstedt, Palle. "Musicking with Algorithms: Thoughts on Artificial Intelligence, Creativity, and Agency." Handbook of Artificial Intelligence for Music. Springer, Cham, 2021. 873-914.

Davidson, Jane W. "Visual perception of performance manner in the movements of solo musicians." Psychology of music 21, no. 2 (1993): 103-113.

Diapoulis, Georgios, and Palle Dahlstedt. (2021). "The creative act of live coding practice in music performance." In PPIG 2021 Doctoral Consortium, University of York, York, UK.

Fanfani, Giovanni, et al. "(Micro-) Performing Ancient Weaving in the PENELOPE project." Performance Research 25.3 (2020): 123-130.

Godøy, Rolf Inge. "Gestural imagery in the service of musical imagery." In International Gesture Workshop, pp. 55-62. Springer, Berlin, Heidelberg, 2003.

Godøy, Rolf Inge, and Marc Leman, eds. Musical gestures: Sound, movement, and meaning. Routledge, 2010.

Jarvis, Ian. "Live coding: sound - gesture - algorithm." In Proceedings of the international conference on live coding. Prado, Madrid (2019).

Keller, Peter E. "Mental imagery in music performance: underlying mechanisms and potential benefits." Annals of the New York Academy of Sciences 1252, no. 1 (2012): 206-213.

Keller, Peter E., and Iring Koch. "Action planning in sequential skills: Relations to music performance." Quarterly Journal of Experimental Psychology 61, no. 2 (2008): 275-291.

Kiefer, Chris. "Approximate Programming: Coding through Gesture and Numerical Processes." Proceedings of the First International Conference on Live Coding, ICSRiM, University of Leeds. 2015.

Leman, Marc. Embodied music cognition and mediation technology. MIT press, 2008.

Luck, Geoff. "Lecture notes in the x-factor in music." University of Jyväskylä, Jyväskylä, Finland (2015).

Magnusson, Thor. "Algorithms as scores: Coding live music." Leonardo Music Journal 21 (2011): 19-23.

McLean, Alex, and Geraint Wiggins. "Tidal–pattern language for the live coding of music." In Proceedings of the 7th sound and music computing conference. 2010.

Nilson, Click. "Live coding practice." Proceedings of the 7th international conference on New interfaces for musical expression. 2007.

Palmer, Caroline. "Music performance." Annual review of psychology 48, no. 1 (1997): 115-138.

Roberts, Charlie, and Graham Wakefield. "Tensions and Techniques in Live Coding Performance." (2018): 293-317.

Sayer, Timothy. "Cognition and improvisation: some implications for live coding." (2015).

Stewart, Jeremy, and Shawn Lawson. "Cibo: An Autonomous TidalCyles Performer." Proceedings of the Fourth International Conference on Live Coding. 2019.

Stowell, Dan, and Alex McLean. "Live music-making: A rich open task requires a rich open interface." In Music and human-computer interaction, pp. 139-152. Springer, London, 2013.

Stowell, Dan, Andrew Robertson, Nick Bryan-Kinns, and Mark D. Plumbley. "Evaluation of live human–computer music-making: Quantitative and qualitative approaches." International journal of human-computer studies 67, no. 11 (2009): 960-975.

Tanimoto, Steve. "Challenges for livecoding via acoustic pianos." In 3rd International Conference on Live Coding. Morelia, Mexico. 2017.

Toiviainen, P. Personal communication (2015).

Wanderley, Marcelo Mortensen, and Nicola Orio. "Evaluation of input devices for musical expression: Borrowing tools from hci." Computer Music Journal 26, no. 3 (2002): 62-76.

Vetter, Jens. "WELLE-a web-based music environment for the blind." Proceedings of the International Conference on New Interfaces for Musical Expression. Birmingham, United Kingdom. 2020.

Paper II

The creative act of live coding practice in music performance

G. Diapoulis and P. Dahlstedt

Psychology of Programming Interest Group (PPIG), Doctoral Consortium.
York, UK. (2021).

The creative act of live coding practice in music performance

Georgios Diapoulis
Interaction Design
University of Gothenburg,
Chalmers University of Technology
geodia@chalmers.se

Palle Dahlstedt
Interaction Design
University of Gothenburg,
Chalmers University of Technology
palle@chalmers.se

Abstract

Live coding is the creative act of interactive code evaluations and online multimodal assessments. In the context of music performance, novel code evaluations are becoming part of the running program and are interrelated to acoustic sounds. Performers' and audience ability to experience these novel auditory percepts may involuntary engage our attention. In this study, we discuss how live coding is related to auditory and motor perception and how gestural interactions may influence musical algorithmic structures. Furthermore, we examine how musical live coding practices may bring forth emergent qualities of musical gestures on potentially equivalent systems. The main contribution of this study is a preliminary conceptual framework for evaluation of live coding systems. We discuss several live coding systems which exhibit broad variations on the proposed dimensional framework and two cases which go beyond the expressive capacity of the framework.

1. Introduction

1.1. Live coding for the composer-programmer

Live coding practice is a well spread performance activity among computer musicians. Since the funding act of the Temporary Organisation for the Promotion of Live Algorithm Programming, which begun with its draft manifesto (TOPLAP, 2005), a community of few live coders has now been seeing a tremendous expansion. In fact, the term live coding seems to be unclear within academic circles. Many believe that the term corresponds to streaming tutorials where professional programmers show best practices on how to "code live". While this may reflect some aspects of live coding, it does not manifest the potential of a novel computing platform.

Live coding in music performance is a multimodal endeavour. Audition, vision, touch and balance are all forming a closely knit whole during performance. All aforementioned sensory cues may engage both the composer-programmer and audience in a multimodal experience. During performance practice the live coder is typically sharing her screen with the audience. This makes the process of live coding a transparent performance act, in which failure is always a possible outcome. In this manner, both the coder and the audience incorporate the generated music as a proxy to form a mental model of the running program. Consequently, the live coder aims to modify the running program on-the-fly, so that novel auditory percepts may involuntary engage our attentional resources (Escera, Alho, Winkler, & Näätänen, 1998). Such novel performance acts that are realized within the context of "show us your screens", are widely used in algorave parties, where the audience is sometimes dancing during the concert (Collins & McLean, 2014). If the live coder fails to evaluate successfully the current code chunk and commit a system crash, she might start all over once again, or if she feels exhausted she might seek for some encouragement from the audience. This gestural communication between the audience and the live coders is well established in live performances as the algoraves are about to become 10 years old in 2022.

1.2. Humans in the loop

Live coding is a novel performance practice and maybe extends the notion of human-centric computing. This is because the human participant has an active role which is determined by the social nature of musical activities (Collins, McLean, Rohrhuber, & Ward, 2003; Thompson, 2015). On the other hand, there are still difficulties how to humanly embody our interaction with algorithms. Musicians are encountered to a first-hand experience of the semantic gap, as this is portrayed between the generated music and the

typed code expressions. The most promiment way, to date, to experience embodied interactions in live coding is taking flesh during a dance performance (McLean & Sicchio, 2014). Even in this scenario, the dancer is carrying on the performance within the predominant absence of causal or direct auditory percepts linked to musicians' gestures. Besides these drawbacks, the world of live coders is a lively and widely divergent community of hackers, expert and novice programmers, academics, professional and hobby musicians and most likely a bunch of retired enthusiasts within the next decades (Nilson, 2007). Furthermore, there is a broad range of computer music conferences that have incorporated live coding as a research topic, but most importantly there is a quasi-annual conference on live coding (ICLC), first appeared in 2015.

1.3. Outlining the purpose of the study

In this study, our purpose is to present a conceptual framework for evaluation of live coding music systems. There is a broad variation of systems and practices among live coders and to the best of our knowledge there is no study to date which examines how music systems may differ to each other. The swedish alter ego of Nick Collins has reflected on different practices and actually proposed a battery of live coding exercises (Nilson, 2007). That was a month long live coding exercitiae carried out and documented with Fredrik Olofsson. In the next section we review literature from music psychology and perception along with studies in live coding and human-computer interaction. The focal point is how gestural interactions are employed in musical interaction design. Following that, we present a preliminary conceptual framework which aims to evaluate live coding systems. Finally, we discuss how such frameworks may flourish the development of novel music systems and we reflect on the possibility of a parallel evolution of performance practices.

2. Live coding: musical activities, music performance, systems and practices

2.1. Musical activities

Musical activities may be divided into three categories: music-making, music listening and musical imagery (see Figure 1). Music-making involves both music composition and music performance. Music listening is the most widespread activity as we are exposed to music many hours per day, even involuntary when drinking a coffee in a coffee shop. Musical imagery is the activity of imagining a melody of a song, a musical gesture and so on. A typical case of involuntary musical imagery are the so-called earworms, which is when a melody is in a person's mind.

Here, musical activities are presented as progressively overlapping categories. We may claim that there is also a progressive engagement in musical activities, starting from the least engagement in musical imagery, following to more engagement during music listening and even more engagement during music-making (Luck, 2015). In that manner, during performance the live coder is employing music percepts towards building progressive levels of engagement. When an audience is attending a concert then the dynamics between audience and musicians is also an engaging experience, where dancing and gestural communication are typically of major importance.

2.2. Traditional and live coding music performance

During a concert, the generated music is heard by both performer and audience, and intersubjective music preferences may vary considerably. Whereas a live coding performance incorporates both visual and auditory percepts, to the best of our knowledge, there are no studies which assess how the visual channel contributes to audiences' appreciation. For instance, in traditional music performance is well established that exaggerated bodily movement biases our visual perception of expressivity, which in return contributes to audience appreciation (Davidson, 1993).

Contrary, in live coding the bodily movement is usually minimal. This is an issue which the live coders have to consider if they would like to tighten the engagement with audiences and co-performers. Although, the visual projections have an important role in the live experience as a whole, it is difficult to make educated assessments due to the broad variety of live coding systems. Thus, in our study we consider only the auditory percepts during performance. Here, music listening is seen as an activity which is linked to music preferences and appreciation of the generated music. The live coder is appreciating

the generated music on-the-fly and this may result to structured code evaluations, which are tested and work properly (see Figure 1). In addition, musical imagery is linked to anticipated percepts which occur when people are exposed to music-like stimuli. During music performance this contributes to planning and involves a sequence of bodily movements which are required when playing a musical instrument (Keller & Koch, 2008). Similarly, in live coding performance, the coder is planning her future actions by trial-and-error of novel code evaluations (Tanimoto, 2017). In that manner, the coder is anticipating the auditory percepts of her actions.

The difference between live coding and traditional music performance is that in live coding the learned associations are not necessarily linked to automaticity in gestural control. Instead, the coder is making progressive levels of abstraction, which may be automated to a certain extent (Nilson, 2007). Automaticity in sensorimotor control, especially in the case of typing, is being unfolded in a later stage when the planning has brought forth some sort of mental model of the novel code structures. Such practices are linked to novelty and creativity, which are interrelated concepts to some extent. Typically, creativity depends on some novelty-related component. When there is mismatch of expectations during a live coding performance, this may result to either failure of execution and possibly a system crash or to novel music percepts.

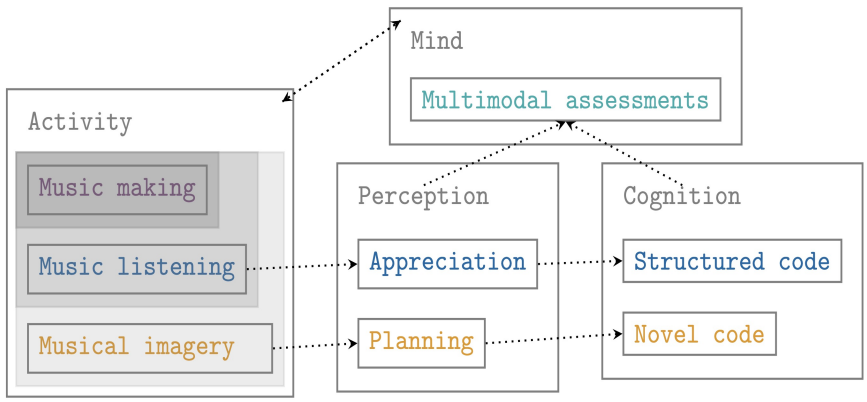


Figure 1 – Musical activities in live coding performance. Description of perception and cognition during performance.

2.3. Perceptual and cognitive aspects of live coding systems and practices

2.3.1. Motor skills in musical interface design

From a motor perspective, a system's response time cannot be smaller to the slowest part of the system (Gibet, 2010). If we transfer this from the motor domain to the user interface design, then we may conclude that the only case of intimate gestural feedback with a user interface (UI) can be achieved based on direct manipulation. Moreover, it is reasonable to assume that in many cases the user's understanding is facilitated when no algorithm is involved during gestural interactions. Contrary, the systems that are used in live coding performances and new interfaces for musical expression (NIMEs), may require a long sequence of gestural interactions which involve algorithmic complexity. The question arises, how algorithmic complexity may be linked to musical gestures as these are portrayed in traditional music performance (Jensenius, Wanderley, Godøy, & Leman, 2010). Do we have to expand the notion of musical gestures (Salazar, 2017)? This can be a plausible argument because musical interfaces share properties from both human-computer interaction (HCI) and traditional music performance. In HCI the gestures in users' interfaces are considerably different than musical gestures. In music performance there is a gestural virtuosicity which is considered a no-go in user interface design.

2.3.2. Multifaceted functionality of musical imagery

During a live coding performance the composer-programmer is making music on-the-fly by incorporating interactive code evaluations. While doing so, she is listening to the music but also imagining anticipated music percepts. The latter is known as anticipatory auditory imagery (Keller, 2012). For instance, in dance music we anticipate the bass drum to be heard on regular intervals within the musical structure. If the composer alternate these regular repetitions of the bass drum our expectations would be mismatched and novel music percepts may arise from such structural modifications. Another significant aspect of musical imagery is that during music listening, pianists have demonstrated activations of involuntary motor imagery (Haueisen & Knösche, 2001). Thus, anticipatory imagery is involved both in action planning and action execution (Keller, 2012).

Here, we stretch the importance of a sequence of gestural interactions and we question how online auditory percepts may influence such mutlilayered gestural unfoldings in live coding. We argue that there should be some sort of mental models that allow the musician to conduct on-the-fly programmatic structures that meet her musical percepts. Interestingly, expert programmers have reported imageries of gestures and other bodily movements during problem solving tasks (Petre & Blackwell, 1999). Whereas direct manipulation can indeed provide a more traditional-like sense of intimate gestural control, more complex systems may also employ gestural imagery. Godøy (2003) sees that contrary to auditory imagery, gestural imagery may be suppressed in time. That is, we can "fast-forward" musical gestures using our mind, whereas the same do not apply to sounds. One cannot compress the duration of a sound and experience similar percepts. How such "gestural compressions" may be related to goal-directed actions? Is the teleological inquiry a mechanism which can facilitate the formation mental models?

If we examine the so-called "earworms" which seem to arise out of circumstances of involuntary musical imagery (notably have been reported widely in non-musicians as well) (Williamson, 2011), then such involuntary actions may trigger the formation of mental models. Such imagery, also known as spontaneous, can be triggered by musical notation. This is known as notational audiation in literature. Live coders employ similar imagery artifacts from chunks of code, as the code is becoming a musical notation (Magnusson, 2011). For instance, the first author is employing visual imagery when performing standard live coding sessions in SuperCollider using the keyboard. This is in the form of geometrical abstractions which are typically realized using low frequency oscillators (LFOs) to manipulate melodic and rhythmic structures that are usually driven by unit generators (UGens).

Several kinds of mental imagery have been reported in both expert and novice programmers (Petre & Blackwell, 1999). For example, both expert and novice programmers reported that they employed visual imagery when structuring a program. Gestures and algorithms can be difficult to put into strict boundaries, that is to put them into segmented structures. On the other hand, auditory and visual percepts exhibit segmentation properties. For instance, a sound event may attribute segmented boundaries to gestures via the onsets and the offsets of the sound. This is how computer music is linked to bodily movement throught its temporal structure (Palmer, 1997).

2.3.3. Knit together systems and practices

So far, we have argued that a blend of effortfull and involuntary imagery takes place during performance. One instance of imagery is immidiately linked to gestural interactions (Godøy, 2003), which has temporal advantages in comparison to auditory percepts (ie. "fast-forward" gestural representations). Moreover, musical notation can function as a source of imagery-induced processes. One can think that there is a solid ground already so to engage with the study of musical gestures in live coding, but the broad variations of systems and practices bring about a plethora of gestural interactions. On top of that, we have to take into consideration some principles of human-computer interaction. For instance, standard live coding systems which are based on typing on a keyboard are known to offer terrible closeness of mapping (Blackwell & Collins, 2005). In fact, live coding systems that incorporate the keyboard for

gestural control may be seen as obscured, as typing on a keyboard is "neither observed nor significant"[1] (Jensenius et al., 2010).

Here, the contribution by Marije Baalman is more than significant (Baalman, 2009). Baalman demonstrated in her "Code LiveCode Live" session that typing on a keyboard can bring about meaning and made the transition from an unspecified and "non significant" domain to the musical gestures domain. This point is likely the very essence of this article, which is linked to the very title of this paper. That is, potentially equivalent live coding systems may bring about different performance practices. This in return, can bring forth novel contributions such as assigning meaning to "non significant" actions, like typing on a keyoard.

3. A conceptual framework for evaluation of live coding music systems based on gestural interactions

Our investigation began by examining different systems and practices in musical live coding. We reviewed half a dozen live coding systems from the viewpoint of how gestural interactions vary across different practitioners. A turning point which made us realize the importance of variations in performance practices was Marije's Baalman "Code LiveCode Live". The interesting characteristic of Marije's system is that it is potentially equivalent to a standard live coding system. Particularly, Marije used SuperCollider language which is commonly used by many live coders. The only difference between a standard live coding system based on SuperCollider and "Code LiveCode Live" is that Marije activated the built-in microphone and other sensors of the laptop while typing. In that manner she used the typing sounds on the keyboard as the raw material of the composition. This action transformed the meaning of typing in Marije's system. Typing on a keyboard cannot be seen as a non significant action neither as not observed. In contrast, typing has now become a musical gesture, which is actually a sound-producing gesture. That is, it is absolutely significant for the production of the sound. This realization, demonstrated how different practices may bring about creative acts in potentially equivalent live coding systems.

In the following section we are discussing the four main systems under investigation. Next, we present a preliminary visualization of our framework. At the end of this section, we discuss two more systems which cannot be represented using our framework. A discussion follows including potential future work and adjustments can be done to fine tune the framework.

3.1. Four systems under investigation

Here, we focus on four idiosyncratic live coding systems by emphasizing on the gestural control. These are *Al-jazzari* by Dave Griffiths, *stateLogic machine* by Diapoulis, *Code LiveCode Live* by Baalman and *CodeKlavier* by Noriega & Veinberg. The motivation was to examine cases where the musical gestures play an important role in gestural control. As such we included typical cases where the keyboard is used as the input interface, but also exotic (Diapoulis & Zannos, 2012, 2014) and metaphorical design cases, like piano performances (Tanimoto, 2017). The aforementioned variations clearly showed that music systems incorporate design metaphors (Wessel & Wright, 2002). A typical case of a design metaphor denotes a computer music system that was developed based on some existing musical instrument. For example, if we map the letters of the keyboard to a MIDI piano this would account as a metaphorical design. In contrast, literal design setups, like the standard live coding systems, are based on a keyboard which may inhibit gestural expression in comparison to playing the piano.

3.1.1. Code LiveCode Live

Marije Baalman approached the tackling issue of embodiment within laptop performance by incorpotating the clicking sounds on the keyboard into her musical composition (Baalman, 2009, 2015). In that manner, Marije accomplished direct sound to be heard during her live coding performance, as she used physical data as audio input (Nilson, 2007). That was indeed a novel contribution, although the practi-

[1] Original quote by Hulteen (1990) state that (p.310) "A gesture is a motion of the body that contains information. Waving goodbye is a gesture. Pressing a key on a keyboard is not a gesture because the motion of a finger on it's way to hitting a key is neither observed nor significant. All that matters is which key was pressed".

calities of such dual-functionality of the keyboard as both a percussive instrument and a typing UI could not establish a "normal paradigm" for live coding music performance. In fact, Marije's apparatus was not aiming to reach this goal. Most likely her novel contribution was indicating self-referential aspects, as they unfold, during performance. If this apparatus was meant to be taken literally as a "standard" for performance, then it would have triggered a parallel co-evolution of novel keyboard setups, UIs and programming languages.

3.1.2. CodeKlavier

In the same direction the CodeKlavier system (Noriega & Veinberg, 2019), demonstrated a novel live coding performance setup by employing the clavier of the piano as input interface. The novel contribution of the "Hello world" performance [2] was that the authors literally executed a hello world program using the piano as input interface. Whereas this may sound as a "dummy" demonstration, the aim was to be a proof of concept. Below we will focus on the fourth revision of the system, also known as CKalculator[3]. If we imagine a one-dimensional space of *design metaphors* (Dahl & Wang, 2010) and *literal design* then the CodeKlavier would be on the one end of design metaphor and Baalman's approach on the other end of literal design. Moreover, the two systems differ on another dimension. Marije's design is agnostic to the significance of keypresses, whereas in CodeKlavier CKalculator design the importance of keypresses is highly significant to structure the code. By *algorithm agnostic* we mean that Marije's gestures do not have any impact on the algorithm itself. The coder is doing as many gestures as she likes, she might also do gestures without any temporal contraints and this has no effect on the algorithmic structure of the program.

3.1.3. Al-jazzari

Contrary to the previous two music systems, Dave Griffiths presented one of the very first systems which approached live coding from a low-level perspective (McLean, Griffiths, Collins, & Wiggins, 2010). Al-jazzari is building on a metaphorical design in which a computer game is used as a notation for live coding. The computational approach relies on evaluating commands from a minimalistic instruction set and the input interface is a gamepad controller. Here, every user's action has a significant impact on the algorithm. This is because a positive edge clock is registering the user's input in real-time.

3.1.4. stateLogic machine

Diapoulis and Zannos (2012, 2014) presented a low-level computational approach to deal with live coding. The users' input is provided on the lowest level of the machine, that is, the bit level. The machine is a combination of two finite state machines (FSM), a counter and a decoder, and the user interface is an automaton itself. In the revised version (Diapoulis & Zannos, 2014), the machine was able to recognize regular expressions and generated a minimal type-3 language, which enumerated seven words. Here, the design is literal as the performer is providing the input using switches. The actions of the performer are absolutely significant to the algorithm and the code precedes to the generated music.

3.2. Dimensional framework

One way to evaluate live coding systems is to rely on a multi-dimensional space. Here, we decided to constrain the proposed framework up to three dimensions. Whereas an orthogonal three-dimensional system can be misleading, we decided to employ such representation to facilitate the understanding of the reader. Our intention was to provide a comprehensible visual representation of the framework. Thus, we engaged in a process of identifying the most important semantic differentials which can reflect the variations between the systems under investigation.

Our first observation was that the *interface design* can be either *metaphorical* or *literal*. We introduce here the term "literal design" to denote that the system fulfils the requirements of a standard live coding system. That is, the interface is based on some sort of electronic components such as switches, keyboards, circuits and the like. This is the first dimension (X-axis) of the framework as shown in Figure 2.

[2] CodeKlavier - hello world (Anne Veinberg playing piano and coding at the same time!) `https://youtu.be/ytpB8FB6VTU`

[3] Anne Veinberg, Felipe Ignacio Noriega - The CodeKlavier CKalculator (...) | Lambda Days 2019: `https://www.youtube.com/watch?v=0fL40oLU8C4`

Here, we assert that the interface design should be reflected to the qualities of gestural interactions.

The second dimension of the framework examines the importance of gestural interactions on the algorithm of the system. For instance, in the case of stateLogic machine every input provided from the user modifies the algorithm of the system. To provide a more concrete example, here, we have to introduce a third dimension which has *direct manipulation* on the lower end and *algorithmic complexity* on the upper end. If we imagine a continuous gesture on a tangible interface then this is a direct manipulation gesture. The question arises, "what if there is an algorithm behind this direct manipulation gesture?" (Björk, 2021). For this reason, the second dimension of the framework clarifies whether the gesture is actually *significant* to the running algorithm or it is *agnostic* to it. Thus, the second dimension, as shown on the Y-axis, corresponds to *gestural mapping* and the third dimension, Z-axis, to *user interaction*.

The directionality of the axes was designed to represent concrete concepts on the lower end and abstract concepts on the upper end. This is a cognitive paradigm that shows a directionality from low-level concepts to high-level concepts. The three basic dimensions on the framework are shown in Table 1. The dimensions represent some of the basic processes that the live coder is engaged with during the development of a musical performance system.

Table 1 – Each axis denotes a process which is represented by semantic differentials. The directionality of the axes is composed by low-level concepts on the lower end and high-level concepts on the upper end.

	Process	Low-level (concrete)	High-level (abstract)
X-axis	*Interface design*	literal design	metaphorical design
Y-axis	*Gestural mapping*	algorithmic significance	algorithm agnostic
Z-axis	*User interaction*	direct manipulation	algorithmic complexity

Finally, as shown in Figure 2 we included a binary dimension as was proposed by Tanimoto (2017). In a live coding system, either the code preceeds the music (code-first) or the the music preceeds the code (music-first). Regarding the case of Baalman's system, we categorize it as a music-first because the typing sounds are feeded forward to the generated music. Here, we have to highlight that if the performer does not execute any commands to switch on the built-in microphone of the laptop, then no typing sounds will be heard. In that manner, the system may also be categorized as code-first. In principle, a more accurate description would be to go beyond the binary division of code-first and music-first to include more categories. In this case, Baalman's system would be a conditional music-first system.

3.3. Beyond the expressive capacities of the framework

Below we present two cases which cannot be represented with the propose framework.

3.3.1. Type-A personality

The piano composition "Type-A personality" was performed during the first international conference in live coding (Collins & Veinberg, 2015). The pianist is playing the piano but also typing on a keyboard at the same time. Indicatively, keyboard characters are shown on the score in the video of the performance. This system cannot be categorized neither as a metaphorical design nor as a literal design. Furthermore, the gestural mapping seems to be significant to the algorithm but an interview with the either the composer or the performer will shed light to it. For example, if a machine listening component performs online music analysis then the gestural interactions are significant to the algorithmic. Finally, the system seems to incorporate only direct manipulation.

3.3.2. Threnoscope

The live coding system "Threnoscope" presented a blend of visual notation coupled to a standard live coding system (Magnusson, 2014). The implementation was done in SuperCollider and the performer can use both the keyboard and the mouse for user interaction. In that manner, the system incorporates both direct manipulation and algorithmic complexity. The gestural interaction is agnostic the algorithm, although when the performer interacts with the visual notation it can adjust numerical values on different

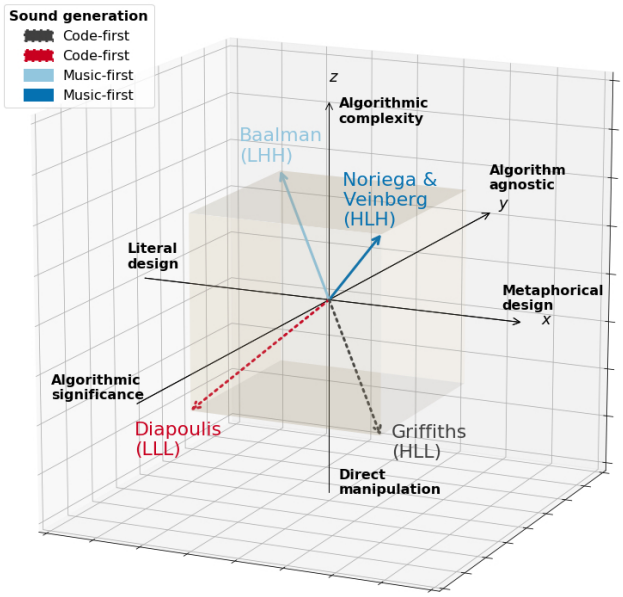


Figure 2 – Dimensional framework from the viewpoint of gestural interactions. Uppercase characters "H" and "L" correspond to high-level and low-level concepts for the triad XYZ axes, respectively. Dashed arrows show systems that the code precedes the generation of sound.

parameters.

4. Discussion

We presented a preliminary version of an evaluation framework for musical live coding systems from the viewpoint of gestural interactions. Musical gestures in traditional music performance have a long history and the musicians are well-known to be experts of sensorimotor control. A central theme in our study was to built upon a theoretical background in which the musical activities are seen as nested categories. Indicatively, music-making incorporates both music listening and musical imagery. An attempt was made to explain how gestural unfoldings may influence our mental model of the running program. This may be explained through segmented structures that are realized by auditory percepts, which in return may influence the fast-forward of gestural interactions. On the level of musical imagery, spontaneous imagery has shown to influence motor activity (Haueisen & Knösche, 2001), thus, it can be involved in action planning and execution (Keller, 2012).

A clear distinction between musical live coding systems and practices is made, to facilitate the understanding of the reader. Interestingly, we presented a case (Baalman, 2009) in which potentially equivalent systems can bring about different performance practices. Our motivation was to conceptualize how variations in performance practices may contribute to the development of novel systems. The preliminary nature of the proposed framework is expemplified by two special cases which cannot be represented in a consistent manner. Furthermore, the three-dimensional representation that was choosen for visual communication, may be misleadning for the reader as the orthogonality of the axes typically corresponds to independent concepts.

Furthermore, we introduced the dimension of gestural mapping from the viewpoint of how gestural interactions may have an effect on the running algorithm. This clarifies the reason that the direct manipulation and the algorithmic complexity are presented as semantic differential concepts.

Future studies should evaluate the validity of the framework, either using quantitative, qualitative or mixed methods. Indicatively, interview studies can be very beneficial for verifying shared conceptions among the community of live coders. How such frameworks may benefit the live coding community? We believe that by offering a conceptual framework based on the viewpoint of gestural interactions we will facilitate the development of novel performance systems. Engaging into iterative processes by practicing and making efforts to go beyond the expressive capacities of such frameworks can be only beneficial for our imagination during performance.

5. References

Baalman, M. (2009). *Code LiveCode Live.* Retrieved 2021-09-29, from `https://marijebaalman.eu/projects/code-livecode-live.html`

Baalman, M. (2015). Embodiment of code. In *Proceedings of the first international conference on live coding* (pp. 35–40).

Björk, S. (2021, September). Personal communication.

Collins, N., & McLean, A. (2014). Algorave: Live performance of algorithmic electronic dance music. In *Proceedings of the international conference on new interfaces for musical expression* (pp. 355–358).

Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2003). Live coding in laptop performance. *Organised sound*, *8*(3), 321–330.

Collins, N., & Veinberg, A. (2015). *"type a personality." a performance at iclc 2015.* Retrieved from `https://www.youtube.com/watch?v=0fX0AymCtgA`

Dahl, L., & Wang, G. (2010). Sound bounce: Physical metaphors in designing mobile music performance. In *Nime* (pp. 178–181).

Davidson, J. W. (1993). Visual perception of performance manner in the movements of solo musicians. *Psychology of music*, *21*(2), 103–113.

Diapoulis, G., & Zannos, I. (2014). Tangibility and low-level live coding. In *Icmc.*

Escera, C., Alho, K., Winkler, I., & Näätänen, R. (1998). Neural mechanisms of involuntary attention to acoustic novelty and change. *Journal of cognitive neuroscience*, *10*(5), 590–604.

Gibet, S. (2010). Sensorimotor control of sound-producing gestures. In *Musical gestures* (pp. 224–249). Routledge.

Godøy, R. I. (2003). Gestural imagery in the service of musical imagery. In *International gesture workshop* (pp. 55–62).

Haueisen, J., & Knösche, T. R. (2001). Involuntary motor activity in pianists evoked by music perception. *Journal of cognitive neuroscience*, *13*(6), 786–792.

Jensenius, A. R., Wanderley, M. M., Godøy, R. I., & Leman, M. (2010). Musical gestures: Concepts and methods in research. In *Musical gestures* (pp. 24–47). Routledge.

Keller, P. E. (2012). Mental imagery in music performance: underlying mechanisms and potential benefits. *Annals of the New York Academy of Sciences*, *1252*(1), 206–213.

Keller, P. E., & Koch, I. (2008). Action planning in sequential skills: Relations to music performance. *Quarterly Journal of Experimental Psychology*, *61*(2), 275–291.

Luck, G. (2015, September). *Lecture notes in the x-factor in music.* University of Jyväskylä, Jyväskylä, Finland.

Magnusson, T. (2011). Algorithms as scores: Coding live music. *Leonardo Music Journal*, *21*, 19–23.

Magnusson, T. (2014). Improvising with the threnoscope: Integrating code, hardware, gui, network, and graphic scores. In *Nime* (pp. 19–22).

McLean, A., Griffiths, D., Collins, N., & Wiggins, G. (2010). Visualisation of live code. *Electronic Visualisation and the Arts (EVA 2010)*, 26–30.

McLean, A., & Sicchio, K. (2014). Sound choreography<> body code. In *Proceedings of the 2nd conference on computation, communcation, aesthetics and x (xcoax)* (pp. 355–362).

Nilson, C. (2007). Live coding practice. In *Proceedings of the 7th international conference on new interfaces for musical expression* (pp. 112–117).

Noriega, F. I., & Veinberg, A. (2019). The sound of lambda. In *Proceedings of the 7th acm sigplan international workshop on functional art, music, modeling, and design* (pp. 56–60).

Palmer, C. (1997). Music performance. *Annual review of psychology*, *48*(1), 115–138.

Petre, M., & Blackwell, A. F. (1999). Mental imagery in program design and visual programming. *International Journal of Human-Computer Studies*, *51*(1), 7–30.

Salazar, S. (2017). Searching for gesture and embodiment in live coding. In *Proceedings of the international conference on live coding.*

Tanimoto, S. (2017). Challenges for livecoding via acoustic pianos. In *3rd international conference on live coding. morelia, mexico.*

Thompson, W. F. (2015). *Music, thought, and feeling: Understanding the psychology of music*. Oxford university press.

TOPLAP. (2005). *ManifestoDraft.* Retrieved 2021-09-29, from `https://toplap.org/wiki/ManifestoDraft`

Paper III

Bottom-up live coding: Analysis of continuous interactions towards predicting programming behaviours

G. Diapoulis, I. Zannos, K. Tatar and P. Dahlstedt

Proceedings of the International Conference on New Interfaces for Musical Expression (NIME).
Auckland, New Zealand. (2022).

https://doi.org/10.21428/92fbeb44.51fecaab

International Conference on New Interfaces for Musical Expression

Bottom-up live coding: Analysis of continuous interactions towards predicting programming behaviours

Georgios Diapoulis, Iannis Zannos, Kıvanç Tatar, Palle Dahlstedt

ABSTRACT

This paper explores a minimalist approach to live coding using a single input parameter to manipulate the graph structure of a finite state machine through a stream of bits. This constitutes an example of bottom-up live coding, which operates on a low level language to generate a high level structure output. Here we examine systematically how to apply mappings of continuous gestural interactions to develop a bottom-up system for predicting programming behaviours. We conducted a statistical analysis based on a controlled data generation procedure. The findings concur with the subjective experience of the behavior of the system when the user modulates the sampling frequency of a variable clock using a knob as an input device. This suggests that a sequential predictive model may be applied towards the development of a tactically predictive system according to Tanimoto's hierarchy of liveness. The code is provided in a git repository.

Author Keywords

NIME, live coding, tactically predictive

CCS Concepts

•**Applied computing** → **Sound and music computing;** Performing arts; •**Human-centered computing** → **Interface design prototyping**;

Introduction

There are two routes to live coding, either a top-down approach which is how the majority of live coding systems work, or a bottom-up approach which is typically centered on idiosyncratic setups [1][2][3]. We provide an implementation of a bottom-up live coding system capable of generating a minimal language based on regular expressions. We examine whether a bottom-up approach to live coding can inform programming behaviours from predictions run on user's gestural interactions. Recent advances in TidalCycles and Extempore, two popular languages for live coding, demonstrate how artificial intelligence (AI) algorithms can build systems suggesting code patterns to the user [4][5]. This category of systems can be seen as a tactically predictive approach to live coding in Tanimoto's hierarchy of liveness [6].

Such research has not been done in the case of bottom-up systems because these are still relatively rare. Motivated by this observation, we present an experimental study in

which we simulate simple gestural interactions (turning a knob, or adjusting a slider) to guide a lexical analysis process based on bottom-up computations. This experiment is based on a hardware as well as a software prototype. The interaction was conducted on the lowest level of information theory, i.e. the level of individual bits. First-person experiences indicate that this prototype affords intentional control to a certain extent. Furthermore, earlier work suggests that interactive exploration is the way to go [7][8].

We present an algorithmic implementation and a git[1] repository along with an exploratory data-driven analysis on the generated sequences. For the data analysis, we conducted a controlled data generation process to examine any statistical dependencies of the system and explore possibilities for future developments. Our aim was to explore the statistical dependencies of the system over continuous gestural interactions, towards a tactically predictive level of liveness [6].

In this paper, we explore an alternative approach in the interpretation of a data stream provided by a continuous controller. Instead of using this stream as continuous control for a parameter [9], we use it as discrete input in an algorithm to modify the graph structure of the algorithm [10]. We propose a radically simplified interface, with far less visual information elements than our original design. The new design, instead of exploiting “experiences of visibility” and “experiences of meaning” explores aspects of interactivity, like fluidity of actions [11].

The goal is to explore the statistical properties of the generated sequences of tokens and examine how to use them in redesigning an interactive musical prototype. The user interface providing input to the machine is composed of a single potentiometer (knob). We explore firstly the outward behavioral characteristics of the machine, i.e. the shapes of streams that it produces in response to shapes of input. That makes possible to devise various scenarios for creating sound based on the output stream.

Related work

Typical live coding systems rely on the level-4 liveness, which is described as “informative, significant, responsive and live” [6]. The level-5 liveness is called “tactically predictive” and presented for the first time in the revision of the hierarchy. An example of a tactically predictive system is the autocompletion mode, a feature of modern text editors. Attempts towards a level-5 in live coding systems were recently presented [4][5]. Our aim is to examine whether a bottom-up approach to live coding may afford interactions which can be classified as tactically predictive.

Continuous interaction in live coding

Previous work on continuous musical interactions in live coding ranges from parameter adjustments [9] to approximate programming [10]. The latter explores binary trees interactively by means of continuous gestural control. Baalman [12], experiments with continuous control in live coding performance and develops an environment called GeCola. Armitage and McPherson [13] present a physical prototype based on a stenotype machine, in which continuous sensory input is used for parameter adjustments.

Low-level computing in live coding

Bottom-up approaches to live coding have been presented since Dave's Griffiths Betablocker [1][2]. This video game system is based on assembly instructions for live coding. Reus [3] presents a more radical approach which intervenes on the motherboard of an iMac computer to rewire its internal computing components. Diapoulis and Zannos presented a hardware prototype along of a modulo-8 function implementation coupled to a variable-length Huffman decoder [7]. In a follow-up study [8] they developed an equivalent interactive software interface and implemented a high-level component capable of generating a regular language. The next step to these developments would be to map the tokens to an instruction set and a corresponding computer architecture. Here, we investigate how to map the 7 tokens to the instruction set presented by Collins [14].

Psychology of programming

The physical and virtual prototypes of our initial design relies on experiences of visibility and meaning [11]. More specifically, all the information was visible, which makes the process transparent, but can also overload the user with information. Here, we discuss how to exchange such decisions with designs that foster interactivity. The decision to limit the interaction to a single knob encourages fluid gestural interactions. Visual components such as the current input value can be replaced with a continuous line representing the knob adjustments.

Methods

Below, we focus on the system design and we present the algorithm description along with one of the main features that enabled the continuous interaction, here called *secondary variable clock*. In the second part of the methods, we describe the data generation and the statistical analysis. Following up on our previous studies [7][8], we

introduce a novel 1-to-1 mapping which enables the use of a single knob to generate a formal language.

Our statistical analysis simulates four main scenarios. These occur out of all combinations of either a smooth and steady gesture or a sudden and unexpected gesture in an ascending or a descending direction. We run our simulations on a faster clock cycle than the clock cycle of the initial prototype, and we select a range which may be realized on a physical prototype. Our initial software and hardware prototypes rely on 0.5 seconds clock cycle for the primary fixed clock.

System design

We employed three main algorithms: i) A fully-connected graph, inspired by a 3-bit counter operating on two's complement, which is the equivalent to the modulo 8 function (see Image 1); ii) a variable length Huffman decoder, implemented as a FSM with three states, equipped with one-hot entropy encoding (combinational logic); and iii) an algorithm which performs lexical analysis based on regular expressions. Inspired by our motivation to generate a bottom-up approach to live coding using continuous gestural interactions, we used the output of the decoder to modify the input of the 3-bit counter. This decision enabled us to overrun the push buttons for registering the input to the 3-bit counter, and employ a single knob providing the input values instead. We applied 1-to-1 mapping of the encoder's output to the input of the counter. From the 8 values (0-7) of a 3-bit word, we use only the four uneven ones (1, 3, 5, 7) to eliminate the possibility of infinite length sequences during the tokenization process.

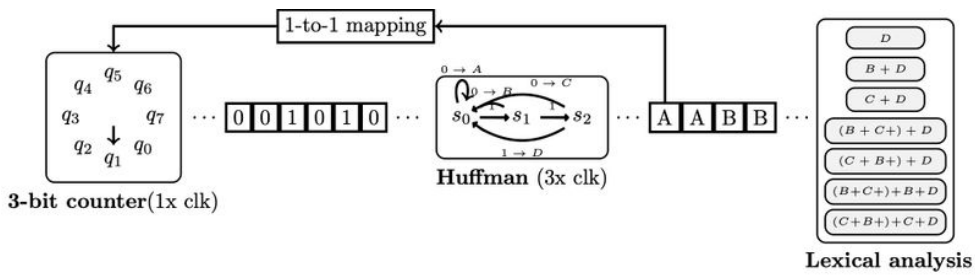


Image 1
High level diagram of the system. From left to right: (i) 3-bit counter, (ii) Huffman decoder, (iii) Lexical analysis.

Algorithm description

Pseudocode

```
Provide input using a knob
    The knob controls the period of a 3-bit counter (secondary variable clock)
        The secondary variable clock may span from 0.5x f_s - 20x f_s
The 3-bit counter is initialized, see (i) Image 1
    The values are registered by the primary fixed clock
        It operates at 1x clk
    Initialization conditions do not radically change the system's behaviour
    Its input is controlled by the 1-to-1 mapping of the encoder's output
        Some symbols may be discarded due to the variable length encoding and the 1x clk
    Only uneven digits are used as input to the counter (1, 3, 5, 7)
        To avoid infinite length sequence in the lexical analysis
    The counter outputs 3 bits at a time
        The output is serialized and fed to the decoder
The Huffman decoder operates at 3x clk (ii)
    The machine operates on variable-length bit streams
    The input to the decoder is 1 bit at a time
    The 3x clk is used to avoid an accumulated stream of bits
    The encoder's output is a stream of symbols
        Feedback channel between the output and the input of the counter (1-to-1 mapping)
    Outputs 4 symbols (A, B, C, D), or bytes
        The encoding is entropic and context-free
The lexical analysis component is serially fed from encoder's output (iii)
    A POSIX expression is generating tokens (words)
    Symbol A mapped to empty string
    The language implementation is noisy
        The input of uneven digits secure a finite length sequence of each token
    The token generation has a variable length of symbols
        List of tokens:
            D
            B+D
            C+D
            (B+C+)+D
            (C+B+)+D
            (B+C+)+B+D
            (C+B+)+C+D
```

Table 1: *The four symbols variable-length encoding and the 1-to-1 mapping to the 3-bit counter input values.*

Table 1

Symbol	Binary code	1-to-1 mapping
`A`	`0`	1 (`001`)
`B`	`10`	3 (`011`)
`C`	`110`	5 (`101`)
`D`	`111`	7 (`111`)

Data generation

The data are generated offline. We simulated four different scenarios of continuous gestural interactions, which can be performed using a knob. These scenarios occur from all combinations of linear and exponential envelopes with either increasing or decreasing values (see Table 2). Together with the four aforementioned combinations, we also examined the effect of different secondary variable clock rates ranging above or below the Nyquist frequency. Specifically, we generated data for six secondary variable clock rates (0.5×, 1×, 2×, 5×, 10× and 20× f_s) and one controlled variable in which we run the simulations based on only the primary fixed clock rate.

Table 2: *Experimental design for the data collection. A design of 2×2 combinations of simulation scenarios for each experimental condition. A total of 7×20 runs were collected for the exploratory data analysis (see an example on Table 3). The rightmost column "Tokens" shows the total number generated across all experimental conditions. The primary fixed clock condition is a controlled variable that is independent of the secondary variable clock. The simulation scenarios are the four possible pairs occurring from combinations between the Gesture type and Envelope type parameters.*

Table 2

Experimental condition	**Gesture type**	**Envelope type**	**Tokens**
Primary fixed clock	-	-	0
Variable clock (0.5×f_s)	ascending/descending	linear/exponential	0
Variable clock (1×f_s)	ascending/descending	linear/exponential	82
Variable clock (2×f_s)	ascending/descending	linear/exponential	645
Variable clock (5×f_s)	ascending/descending	linear/exponential	1190
Variable clock (10×f_s)	ascending/descending	linear/exponential	1462
Variable clock (20×f_s)	ascending/descending	linear/exponential	1007

We generated 20 different sequences for each experimental condition as shown in Tables 2 and 3. Each sequence simulates an actual duration which may be applied in the context of the interactive prototype ranging from 6.25 to 125.0 seconds. If we

assume that the typical duration of the clock cycle is 0.5 seconds, this corresponds to a minimum of 12 clock cycles in the interactive prototype. The generated sequences of tokens have a variable length as indicated in the rightmost column of Table 3. The sampling frequency during the offline data generation was 50Hz and the duration of the experiment has a simulation range 0.25 - 5.0 seconds, with a step of 0.25 seconds. The simulated envelopes have a lower boundary $0.01 \times f_s$, whereas the upper boundary is determined by the experimental condition.

Image 2 shows the data generated by the counter, under the four simulation scenarios of linear/exponential envelopes and ascending/descending direction of movements (gestures), when the output from the Huffman decoder modifies the increment value of the counter.

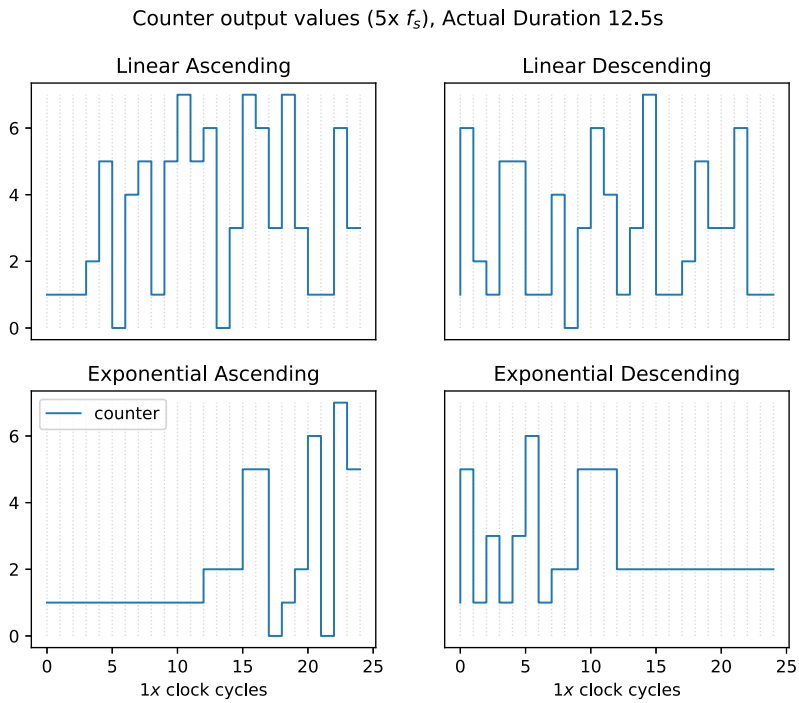


Image 2
Output values of the counter are shown for the four simulation scenarios.

Results

Each of the four simulated scenarios (ascending/descending gestures, linear/exponential envelopes) produces outputs with distinct characteristics independently of the length of the gesture. The statistical analysis identified a relationship between the type of envelope (linear vs exponential) and the differences in the characteristics of the output, which is the number of tokens per second produced. This can be a valuable insight for redesigning the interactive prototype. Finally, the individual token frequencies do not show large variations across experimental conditions and simulation scenarios.

Table 2 shows that the experimental conditions of the primary fixed clock and the secondary variable clock at $0.5\times f_s$ did not produce any tokens. We examined all remaining experimental conditions ($1\times$, $2\times$, $5\times$, $10\times$, $20\times f_s$). We excluded $1\times f_s$ from the analysis as it produced very few tokens across the simulation scenarios, and would be impossible to use for interactive experimentation. From the three remaining conditions of $5\times f_s$, $10\times f_s$ and $20\times f_s$, the exploratory analysis indicates that both the $5\times f_s$ and $10\times f_s$ are more informative experimental conditions for interactive experimentation. This is indicated in the variability of the boxplots in Image 3, as for the case of $20\times f_s$ the corresponding simulated scenarios demonstrate overlapping distributions.

Image 3 (boxplots) shows how the token generation differs based on different experimental conditions and simulation scenarios. It indicates that the simulated scenario $5\times f_s$ can be a better option for the development of a tactically predictive system. This is because there is a clustering of values for the families of linear and exponential envelopes. In practice, this can be implemented when the live coder is overwhelmed with too many token generations. That is, a sudden and unexpected gesture will most likely result in an average token generation of 0.1 tokens per second, whereas in the case of a linear envelope the token generation occurs approximately 0.3 times per second. Thus, by employing a sudden gestural interaction, the user may generate one token every 10s, instead of one token every 3s for the case of a smooth and steady gesture. This system's affordance will provide some time to the user to examine how to proceed to the next token generation, as this would be realized in a 20 clock cycle on the interactive prototype.

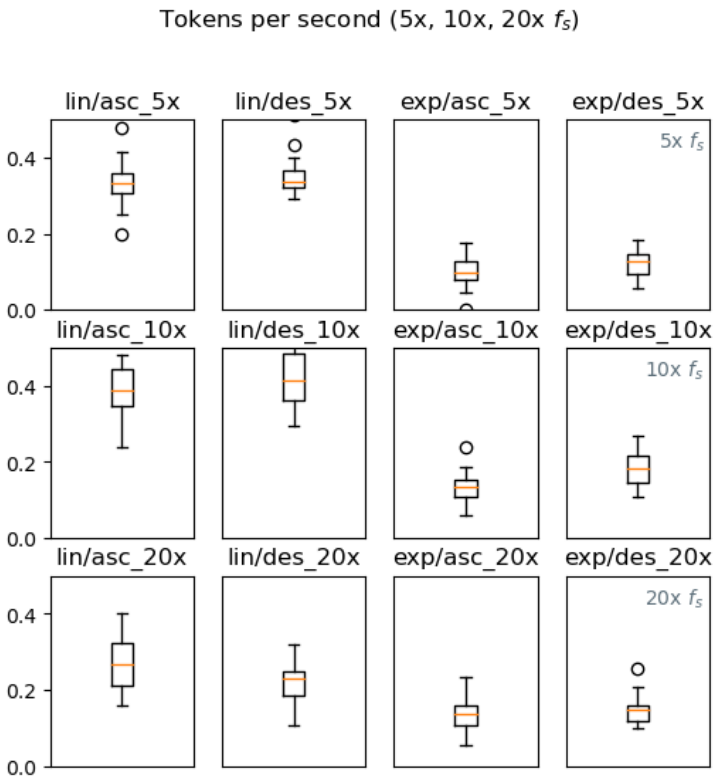


Image 3
Boxplots showing the number of generated tokens per second for three different experimental conditions (5x, 10x, 20x fs) and simulation scenarios ('lin' for linear and 'exp' for exponential envelopes; 'asc' for ascending and 'des' for desceding gestures).

Generated sequences for simulated scenarios

Table 3 shows the length of the generated tokens for the experimental condition of $5\times f_s$. The second column denotes the actual duration in seconds assuming a clock cycle of 0.5s. We selected lower end 6.25s for the actual duration to avoid zero-length sequences, while considering a reasonable minimum time span for applying effortful computations. Our short-term musical memory has an upper time-span of 3-5s [15], and thus 6.25s is close enough for expert user interaction.

Table 3*: Number of tokens for the 5×f_s experimental condition. The columns show the results for the 20 runs for each simulation scenario.*

Table 3

Run (sequence)	Actual Duration (s)	Linear Ascend	Linear Descend	Exponential Ascend	Exponential Descend
1	6.25	2	2	1	1
2	12.5	6	4	1	1
3	18.75	6	7	1	2
4	25.0	5	10	0	2
5	31.25	13	16	3	5
6	37.5	14	13	5	6
7	43.75	11	17	2	8
8	50.0	15	18	5	3
9	56.25	20	19	8	6
10	62.5	19	20	7	8
11	68.75	21	20	6	4
12	75.0	31	22	9	9
13	81.25	27	28	5	11
14	87.5	25	26	6	11
15	93.75	33	34	9	9
16	100.0	33	32	9	13
17	106.25	34	31	16	12
18	112.5	40	42	14	15

19	118.75	44	40	14	17
20	125.0	47	40	22	19

Frequencies of tokens

Table 4 demonstrates that the frequencies of the generated tokens do not show variations across different simulation scenarios.

Table 4: *Frequencies of tokens. Column 1 shows the simulation scenarios; 'lin': linear envelope, 'exp': exponential envelope, 'asc': ascending gesture, 'des': descending gesture.*

Table 4							
$5\times f_s$	D	B+D	C+D	(B+C+)+D	(C+B+)+D	(B+C+)+B +D	(C+B+)+C +D
lin/asc	0.26	0.27	0.07	0.13	0.09	0.15	0.02
lin/des	0.31	0.26	0.07	0.12	0.11	0.10	0.03
exp/asc	0.31	0.20	0.08	0.10	0.07	0.20	0.03
exp/des	0.34	0.24	0.09	0.12	0.12	0.06	0.02

While being simplistic and noisy, this regular language implementation already affords an easy manner to make predictions. When the first symbol of a token sequence is being processed and is either a B or a C, then the search space of the expected tokens is immediately reduced to 3 out of 7 possible outcomes. This observation implies an inherent notion of a "tactically predictive" system, as the programmer can be presented with the future possibilities, although without having immediate control to select any of them.

The predictability was tested subjectively by the first author through experimentation, who concluded that it was relatively easy to predict which tokens will be excluded by the system.

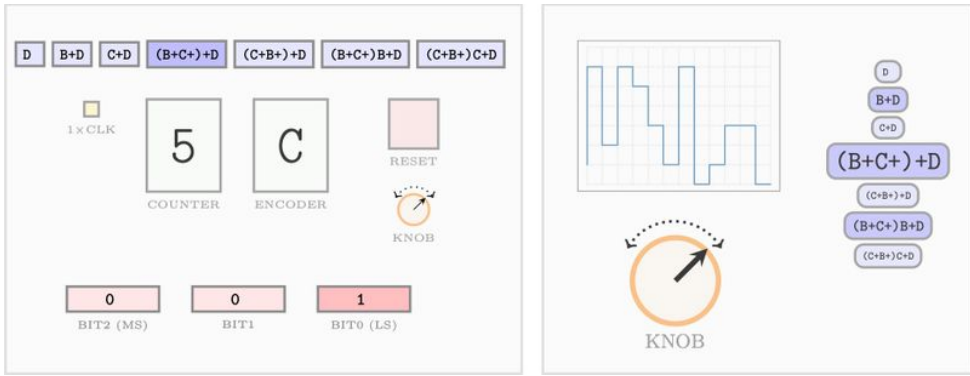


Image 4
Left-hand side shows the GUI from the initial prototype [8]. Right-hand side shows a preliminary redesign, where the token displays may dynamically change in size and color to indicate probabilities for the next token.

Discussion

We presented a bottom-up live coding system and identified its affordances potentially presenting embodied meanings. We examined four pairs of simulation scenarios, as these occur out of the combinations of ascending and descending gestures, which may be either smooth and steady (linear envelope) or sudden and unexpected (exponential envelope) movements. Statistical analysis showed that two clusters are formed across the linear and exponential envelopes. This indicates that a tactically predictive model may rely on such ground truth knowledge. Moreover, a tactically predictive bottom-up system should not only predict the next token, but also provide an estimate of when this token will be produced. Finally, continuous interactions may also carry meaning in the form of embodied metaphors [16]. Such human universals are particularly important when motivated by inclusion and accessibility in design.

We examined aspects of the interactive prototype and we indicated that a redesign should be based on fostering aspects of interactivity of the interface such as fluidity of actions (see Image 4). A detailed analysis of the interface should be carried out based on the patterns of user experience framework [11]. While such endeavour goes beyond the scope of the study, many different aspects may be affected, like the clock period, visual displays and more. Different formal language implementations may also be examined so as to compensate for the noisiness of the proposed regular language used here. This implementation can take advantage of previous developments by mapping

both the tokens and the continuous knob values to an instruction set used for interactive musical experimentation [14].

Conclusions

We conducted a data-driven statistical analysis based on controlled simulations of simplified continuous gestural interactions to examine whether a tactically predictive module for a bottom-up approach to live coding can be implemented. Early evidence suggests that different gestural manners can modify the rate of change of the lexical analysis process using a single knob as input device. Future work is required to analyse dependencies between the variable length sequences of tokens and how these are related to the counter values and envelope shapes, to assess the predictive potential of this system.

Ethical statement

The present study follows ethical principles of open and free software along with low consumption on computational resources. No human participants were recruited on this study. Also, the proposed version for redesigning the hardware and software prototype is considering accessibility in design. Admittedly this ability depends on a deep familiarity of the inner mechanism of the system, which requires solid grounding in the theory of finite state machines and digital design. However, the authors believe that this is a first step towards making such systems more generally accessible. The authors report no potential conflict of interest. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program – Humanities and Society (WASP-HS) funded by the Marianne and Marcus Wallenberg Foundation and the Marcus and Amalia Wallenberg Foundation.

Footnotes

1. https://github.com/gewhere/bottom-up-live-coding ↩

Citations

1. Bovermann, T., & Griffiths, D. (2014). Computation as material in live coding. *Computer Music Journal, 38*(1), 40–53. ↩
2. McLean, A., Griffiths, D., Collins, N., & Wiggins, G. (2010). Visualisation of live code. *Electronic Visualisation and the Arts (EVA 2010)*, 26–30. ↩

3. Reus, J. (2011). *iMac music*. Retrieved from https://jonathanreus.com/portfolio/cmmbe/ ↩

4. Attanayake, U., Swift, B., Gardner, H., & Sorensen, A. (2020). Disruption and creativity in live coding. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 1–5). ↩

5. Wilson, E., Lawson, S., McLean, A., Stewart, J., & others. (2021). Autonomous Creation of Musical Pattern from Types and Models in Live Coding. ↩

6. Tanimoto, S. L. (2013). A perspective on the evolution of live programming. In *2013 1st International Workshop on Live Programming (LIVE)* (pp. 31–34). ↩

7. Diapoulis, G., & Zannos, I. (2012). A minimal interface for live hardware coding. *Live Interfaces*. ↩

8. Diapoulis, G., & Zannos, I. (2014). Tangibility and low-level live coding. In *ICMC*. ↩

9. Magnusson, T. (2014). Improvising with the Threnoscope: Integrating Code, Hardware, GUI, Network, and Graphic Scores. In *NIME* (pp. 19–22). ↩

10. Kiefer, C. (2015). Approximate Programming: Coding through Gesture and Numerical Processes. In *Proceedings of the First International Conference on Live Coding, ICSRiM, University of Leeds*. ↩

11. Blackwell, A. F. (2015). Patterns of user experience in performance programming. In *Proc. First International Conference on Live Coding*. ↩

12. Baalman, M. (2020). the machine is learning. *Live Interfaces*. ↩

13. Armitage, J., McPherson, A., & others. (2017). The Stenophone: live coding on a chorded keyboard with continuous control. *International Conference on Live Coding*. ↩

14. Collins, N. (2015). Live Coding and Machine Listening. In *Proceedings of the International Conference on Live Coding* (pp. 4–11). ↩

15. Snyder, B., & Snyder, R. (2000). *Music and memory: An introduction*. MIT press. ↩

16. Lakoff, G., & Johnson, M. (1980). Metaphors we live by. Chicago: Univ. *Press, Chicago/IL*. ↩

Paper IV

Livecode me: Live coding practice and multimodal experience

G. Diapoulis

Proceedings of the 33rd Annual Workshop of the Psychology of Programming Interest Group (PPIG).
London, UK. (2022).

Livecode me: Live coding practice and multimodal experience

Georgios Diapoulis
Interaction Design
Chalmers University of Technology,
University of Gothenburg
georgios.diapoulis@chalmers.se

Abstract

I present a practice-led research design to explore relations between listening and non-listening conditions during a month-long live coding practice. A documentation of the live coding sessions and textual data from my daily diaries are presented in a git repository. The study offers a set of observations related to musical and programming practices, an ongoing work on a visual helper and outlines issues related to solo live coding practice.

1. Introduction

A central component of musicianship is diligent practice. Musical practice is a daily activity that musicians do, and the same applies to live coding. Musical live coding uses on-the-fly computer programs to generate sound patterns (Collins, McLean, Rohrhuber, & Ward, 2003). It can also be seen as an approach to experience composition while composing musical outcomes (Sorensen & Brown, 2007), and it is an improvisational practice. A significant difference between learning how to live code and learning how to master composition or instrumental music performance is that there is no school to learn how to live code, except a wiki page on TOPLAP website [1]. Learning by example and trial-and-error problem-solving technique is at the heart of every live coding practice. Click Nilson, a persona of Nick Collins, contributed the first study on live coding practice and proposed a set of exercises for improving coding and musical expertise (Nilson, 2007). He addressed these topics by consulting developmental and educational psychology literature studies and carried out a month-long daily practice with Fredrik Olofsson.

My perspective was to carry out daily practice sessions, having in my mind how a novice user would explore live coding. The primary motivation was to improve my musical live coding skills and explore the role of music listening during live coding practice. To explore the role of listening in the context of multimodal perception, I conducted daily sessions with listening and without listening to the generated sounds. The latter was done by muting the soundcard's audio output to the speakers. Listening to the sound while coding is an indispensable part of live coding practice (A. F. Blackwell & Collins, 2005). I posed the question, what if the live coder does not listen to the sound? Thus, I experienced whether listening to the generated sound patterns may help me to understand the written code and benefit my live coding practice.

For the present study, I carried out daily solo live coding sessions. At times it felt like a conversation with myself (Gamboa, 2022), but also a familiar thing to do as I have been practising musical instruments for many years. For the live coding sessions I used SuperCollider [2], a programming environment for sound synthesis and algorithmic composition, and documented temporally accurate representations of the coding sessions. Every day I conducted both listening and non-listening sessions, and I wrote short diary entries at the end of the day, sometimes listening the sessions afterwards by replaying the code recording.

Multimodal experience in live coding studies has been approached from an audience perspective (Burland & McLean, 2016). In this article, I focus on the subjective experience of the live coder. Contrary to Burland and McLean (2016), who discuss how audio-visual information facilitates audience

[1] https://toplap.org/wiki/Live_Coding_Practice
[2] supercollider.github.io/

aesthetic experience, my focus was how audio-visual information could be useful for the performer. Thus, instead of focusing on audience appreciation and enjoyment, I focus on the relation between the auditory and visual percepts of the live coder. How can I get informed when I do not listen to the musical outcome? For that purpose, I explored how visual information from the audio spectrum analyzer, a visual representation of the magnitude of a signal as a function of frequency, can be helpful for the performer.

This study is focused on reproducing the methodology of the seminal practice sessions by Collins and Olofsson and is also influenced by the methodological approach presented by Blackwell and Aaron (2015). It is a practice-led research approach, which can be seen as an extended concept of research through design. By focusing on the four elements by Blackwell and Aaron (i.e., identifying design exemplars, critical orientation, exploratory implementation, and reflective assessment), I contribute a code repository of the coding sessions and reflections related to musical and programming practices, multimodal perception and general issues related to the activity of practising live coding alone.

2. The practice of musical live coding

The computer can be seen as the 'natural tool' of electronic music (Nilson, 2007). During a live coding session, the code is translated to musical outcomes, and the musicians constantly listen to the generated sounds (A. F. Blackwell & Collins, 2005). Collins presented a month-long live coding practice session along with Fredrik Olofsson. The practice sessions were conducted in SuperCollider, and the documentation of the self-administered daily sessions of unaccompanied solo practice is available online (`https://swiki.hfbk-hamburg.de/MusicTechnology/819`). During this month-long practice, the two live coders had different approaches. Collins had a daily plan and exercised specific algorithmic problems, like the 3x + 1 problem, whereas Olofsson did not have a specific plan per day. The sessions were 'blank slate', meaning there was no pre-written code to be executed. The goal of both coders was to practice one hour per day. That is, one hour-long practice sessions simulating a performance setting. Collins sums up his contributions on three main aspects: (i) isolation exercises, (ii) connectivity exercises, and (iii) repertoire implications. Isolation exercises are activities that a live coder can carry out alone and do not necessarily relate to musical practice. Examples include practising fast typing or solving mathematical problems. Connectivity exercises are music-related and address issues the live coder has to confront during a live session. These connectivity exercises may include controlling musical tension, mixing audio signals, and so on. The third aspect of, what I call, repertoire implications addresses issues such as code sharing in laptop ensembles, among others.

Sorensen and Brown (2007) report five computational techniques used during live coding practice. These techniques are generic and can be extended to different aspects of electronic music, like using probability functions, periodic functions and modulo arithmetics, among others. They elaborate on the programming practices during live coding, such as code expansion, function abstractions and keyboard shortcuts. Collaboration and communication are two live coding practices that are also essential. Collaborative live coding has also been a risk management technique (Roberts & Wakefield, 2018). In my month-long practice, I did not do any collaborative sessions. On the communication aspect, I communicated my daily practices with various people, from academics to practitioners and non-musicians. However, I feel I should have been more systematic in this.

Magnusson (2015) explored notational aspects of live coding practice. Magnusson identifies a difference between prescriptive and descriptive aspects of visual notations and offers a solution to the problem of making a connection between code representations and temporal representations of musical events. Another approach to visualisation of musical forms have been recently presented (Dal Rì & Masu, 2022), either as a linear temporal evolution or as clusters showing event density per family of sounds. The authors suggest that these two visualization techniques can be complementary to each other, and reflect on issues related to attention span between coding and visual representations of musical form.

2.1. Psychological aspects of live coding practice

The present study offers a perspective in which the user's task is unlike the 'normal model' of live coding practices (A. F. Blackwell & Collins, 2005). The normal model of musical live coding involves listening to the sound while simulating a performance setting. Still, I conducted at least one session daily without listening to any sounds. Consequently, it contributes to the literature with a use case where the user's needs are unlike the needs of a live coder. In that manner, I present an unusual case of live coding practice that is unlike previous research in the field. The outcome of this study is compiled by examining the reflective diaries, and I discuss how the non-listening condition can benefit or hinder a live coding session.

3. Design of the study

The design of the study is simulating a performance setting, similar to Collins and Olofsson. By performance setting, I refer to a continuous live set with a predetermined minimum duration and a continuous evolution of sound patterns. There was no systematic preparation before each session, and in most cases, no preparation. All practice sessions were 'blank slate' live coding, that is, with no use of pre-written code. That was my main design exemplar, and I conducted a pilot and an experiment proper (A. F. Blackwell & Aaron, 2015). The study was conducted in two parts, a pilot study took place in August 2022 and a proper one in October 2022. A series of 12 daily live coding sessions were carried out during August, and a month-long daily practice was carried out in October. The experimental designs were different as an outcome of the post-proceedings policy of the PPIG conference.

Part of the critical orientation (A. F. Blackwell & Aaron, 2015) is reflected in the experimental design which was influenced by the seminal study of Davidson (1993) on the perception of expressive performance. This critical orientation led me to question how music listening is useful to the performer. Davidson (1993) designed a study in which she assigned expressive manners to violinists and recorded both video and audio recordings. Later, for validating the expressive manners Davidson conducted a perceptual experimental showing stimuli of full-body movement from the violin performances. Part of the methodological novelty of the study was that the stimuli were based either on visual information only, audio information only, or showing both audio-visual information from the violin performances. In that manner, Davidson controlled the perceptual channels demonstrating that the visual channel can bias our perception of expressivity in music performance. Here, I applied a similar orientation and designed a listening and a non-listening condition, but without assigning any expressive manners or specific tasks during my practice.

During the pilot practice sessions in August, I did three daily sessions of 500 seconds each (36 independent sessions in total). This is approximately 8 minutes, considered a short duration for a live coding performance. The three conditions of the August daily sessions are coded in the file names of the practice sessions as 'No-Audio Level meter' (NAL), 'No-Audio Spectrum' (NAS), and 'Audio-Visual' (AV) conditions. Both NAL and NAS sessions did not include any audio, as I muted the audio from the sound card to the loudspeakers. The NAL sessions were conducted without audio from the loudspeakers, and the only informative cues about the generated sound patterns came from a stereo sound level meter (see Figure 1). The NAS sessions were also conducted without any auditory cues, and the visual cues included both a sound level meter and a spectrum (see Figure 2). During the AV sessions, I could listen to the generated sound patterns and see the visual cues coming out of the sound level meter and the spectrum (see Figure 2).

During the pilot study in August, I focused on a controlled experimental design, and I also aimed to control the listening and acoustic conditions. By controlled conditions, I refer to designing a daily plan about the order of the listening conditions (NAL, NAS, AV), but also to control the sound levels from the computer to the listener. In that manner, I used a MacBook Pro laptop, reproducing the generated sounds from the built-in loudspeakers on maximum levels. I realised that this experimental setup and a controlled experimental setup could be challenging to provide fruitful results for my study. After the PPIG conference and my presentation, I received valuable feedback, which made me redesign the ex-

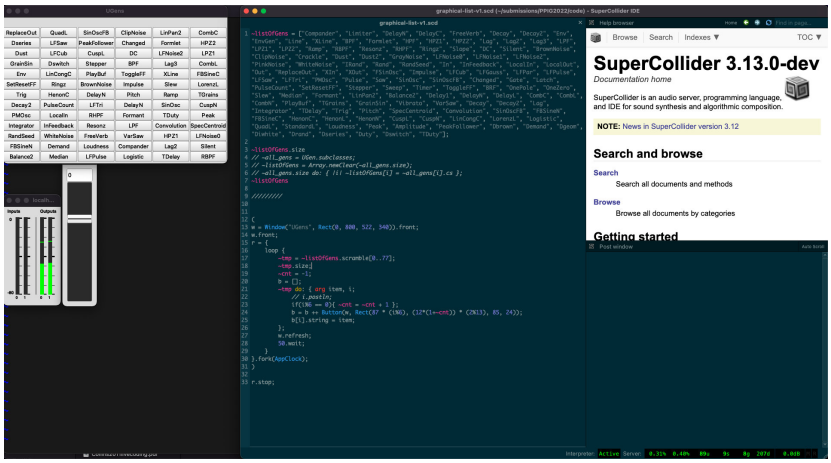


Figure 1 – Pilot experimental setup in SuperCollider for the NAL condition.

perimental setup. The main change was to discard the NAL sessions, as I found them uninformative and maybe annoying. That phase can be seen as the exploratory implementation (A. F. Blackwell & Aaron, 2015). During the experiment proper in October, I did not attempt to control the acoustic conditions. Thus, when I refer to listening to the sound output, I may interchangeably refer to listening from loudspeakers or headphones.

During the month-long experiment in October, I conducted two sessions (AV and NAS) per day of at least 1500 seconds each, by swapping the order of the first and second session every day. This is equivalent to two 25 minutes sessions per day, which is considered an adequate duration for a live coding performance. I manually monitored the duration of every session, and I did not set a hard limit on finishing the sessions. Contrary, during the August sessions, I had programmatically set a hard limit to finalise the sessions after the 500 seconds time limit. For the pilot study in August, every session was catalogued with an audio recording and a timestamped text file showing every code execution. For the experiment proper in October, the audio recordings were not conducted, mainly due to large allocations of memory storage.

The code was captured using the `History` class of SuperCollider. This class can provide reproducible live coding sessions in terms of code executions and audio output. The collected data are timestamped scripts, in the format of plain text files (`*.scd` files), for SuperCollider.

Short-length reflections were written about the daily sessions. The length varies but is no more than 200-300 words per day. Some daily diaries are as short as 1-2 sentences, especially during the October sessions. It is difficult to distinguish whether they can be considered reflective diary entries or note-taking prompts. Complete documentation of the code and the diaries is provided online (`https://gitlab.com/diapoulis/livecodeme/`).

3.1. Visual helper to aid creativity

Part of the experimental design includes a graphical interface (GUI) as a visual helper, as shown in the top-left of Figure 1 and Figure 2. It is a simple helper device that emulates using Post-it notes on the screen (A. Blackwell & Green, 2003). My motivation was to have some visual aid that shows various unit generators (UGens), and get inspiration when building sound synthesis engines. A UGen is the basic building block for sound synthesis engines in many programming languages. SuperCollider has a plethora of UGens, either in the main library or as third-party developments, which makes difficult to recall each one of them. My idea was to begin from a visual helper which could potentially be developed into a software agent. As shown in Figure 1, the initial implementation was done by simply

randomising a list of UGens and printing them on a GUI. Initially, I used a dense matrix format, with dimensions 13x6, refreshing the GUI every 50 seconds. During the pilot study, I selected a constrained set of UGens to be used across the study. The initial list had 128 entries, and I manually selected basic signal generators (like sine and noise oscillators), routing UGens, filters, triggers and envelopes. I found this family of UGens to be limiting my coding practice.

During the study, the GUI did not change significantly. I kept its main passive functionality, and I reduced drastically the amount of UGens shown at a time. During the experiment proper the GUI was showing 8 UGens at a time, and it was updated every 30 seconds (see Figure 2). Some preliminary research on how this can be developed into a software agent was done and is discussed across the diaries. Some of the early conclusions came out of the pilot study, which was to transform the GUI to a 'disruptive' software agent (Attanayake, Swift, Gardner, & Sorensen, 2020), that induces or simply replaces code segments with UGens of similar functionality. This part of the study is still a work in progress.

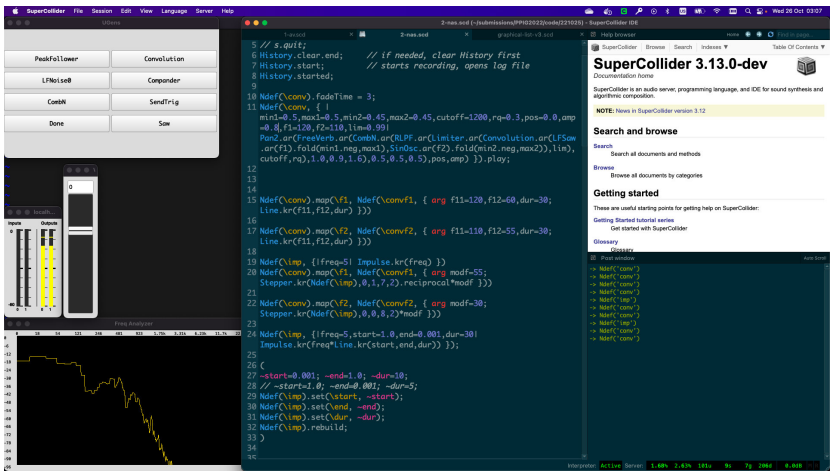


Figure 2 – Experiment proper setup in SuperCollider for the NAS and AV conditions.

3.2. Overall comments on the experimental setup

After my initial attempt to do a controlled study, I feel that I shifted to the other end of having no constraints set in place. Initially, I was inspired from the approach by Fredrik Olofsson while doing the month-long practice sessions with Nick Collins, and did not have a specific coding daily plan. Later I realised that Fredrik Olofsson had also imposed a set of constraints to another practice session that conducted with Marcus Fjellström [3]. In these sessions Olofsson used a single synth definition across all sessions. That is radically different from my experimental design, where I experimented with coding synth representations for sequencing, sound generation, control structures, machine listening among others.

4. Looking into the sound

The design of the study affords an alternative view to live coding. While playing music without listening to it sounds like a no-go, this is the main contribution of this study. Thus, I here present a reflective assessment compiled from the daily diary entries (A. F. Blackwell & Aaron, 2015).

4.1. Reflections from experience

It goes without saying that musical live coding without listening to the sound can hardly be a rewarding activity. Musicians use sound as the primary informative cue during a performance, and I have never

[3] https://fredrikolofsson.com/f0blog/pact-april/

attended a concert where the performers intentionally choose not to listen to the sound. During my non-listening practice sessions, I was sometimes curious to hear what the musical outcome sounded like, and I did sometimes unmute momentarily during the pilot study. During the experiment proper, I did not unmute the non-listening condition, instead, I listened to some of the code recordings after the end of the sessions. It is out of the question that when you live code and do not listen to the sound there are limited possibilities of what can be perceived from the musical outcome. For instance, if I program a sine oscillator and assign a specific frequency, I need perfect pitch to be able to 'hear' how the outcome may sound like. All in all, the only informative cue when non-listening to the sound can be reconstructed by imaging the musical outcome. Musical imagery is a valuable musical ability but it may be a less engaging activity in comparison to music listening. That is, we can be really engaged when listening to the music, and this may be expressed with overt bodily movements, such as dancing, whereas it is less likely to start dancing when imagining a melody of a song.

While live coding does not include any direct involvement between the sound energy and the performed actions, when listening to the sound we may be able to simulate sound actions (Jensenius, 2007, p. 19). Even if there is no direct link to any profound sound actions from daily experience or if the sounds are completely synthetic, we are still able to dance to the beat and produce synchronised bodily movements. The same cannot apply to visual percepts, as we generally perform better in audio-motor synchonisation tasks in comparison to visuo-motor tasks (Hove, Fairhurst, Kotz, & Keller, 2013). Thus, there is an embodied understanding that makes the sound and vibration an indispensable part of music making.

4.2. Reflections from the diaries

In the diaries, I found mixed feelings about the non-listening condition. It looks like I both appreciated some surprisingly appealing musical outcomes but was also annoyed when I had no idea what this may sound like. Several musical tasks are hard to do when non-listening to the sound. I found out that I often forget to apply any panning in non-listening conditions. In principle, many connectivity exercises (Nilson, 2007) can be almost impossible to control when non-listening to the sound. Certain aspects like having a percept about the tempo and the beat are impossible to extract by the visual information on the spectrum, during the non-listening condition. Furthermore, it is quite impossible to track for how long a specific musical pattern is active and produces repetitions, and difficult to understand the length of the musical patterns. I noticed that typically in the non-listening condition, the generated sound pattern had a short duration compared to the listening condition.

When listening to the sound, I sometimes experienced a feeling of performance anxiety. I feel that this can hardly be the case when non-listening to the sound. Furthermore, it felt like I spent more time and was more careful when listening to the sound. It seems logical as any programming mistakes can have significant differences in the produced sound levels, which can cause hearing damage. Thus, it looks like that the listening condition is linked to more careful actions, whereas the non-listening condition produces greater extent of experimentation and trial-and-error practices.

4.3. Psychology of programming when non-listening

Whereas I will not present a detailed analysis on the cognitive dimensions of notation between the listening and non-listening conditions, I will present certain aspects that may improve the quality of discussion (A. Blackwell & Green, 2003). I realised that *progressive evaluations* could be complex when non-listening to the musical outcome. When executing a new code chunk, it can be hard to spot differences from the spectrum. For instance, perceiving differences on the spectrum can be difficult when the musical modifications have slight frequency variations. Consequently this can cause *hard mental operations*, as the console may not show any programming errors. Furthermore, I have noticed that when I listen to the sound, I am more careful in progressive evaluations. That may indicate that the listening condition increases *viscosity*, that is resistance to change, but I think that the non-listening is more likely to cause such imbalance. This is because, *error-proneness* is also increased while non-listening to the sound which may cause *premature commitments*. For instance, I may execute a series of *progressive evaluations* and later realise I did no changes on the running program. Because of my inability to identify any differences on the spectrum, I may think that the progressive evaluations impacted the

generated sound patterns. All this description may only apply to me and amplified to some extend by my obsessive commitment on using only the `Ndef` class for sound synthesis in SuperCollider. Further, my obscure coding practice with long one-liners can also cause severe problems to progressive evaluations, error-proneness and hard-mental operations.

5. Discussion

One of my goals is to motivate other live coders to collaborate often and practice in a daily fashion. Because of the very nature of live coding, as a practice which differs substantially from traditional music performance, I have the feeling that live coders do not practice in a daily fashion (at least, this is the case for me). Carrying out a month-long practice made me realise different things, ranging from my programming skills with my favourite programming language, SuperCollider, to how musical agents can be used in live coding and what programming habits I have when live coding.

The present study began with a view more akin to a controlled experiment and developed into a first-person study based on research through design and reflective diaries. It is important to notice that while I keep an interchangable order between listening and non-listening conditions (AV and NAS) in October I cannot say whether this had any impact on the study. I did learn several things about my live coding practice, and first and foremost that I do not practice as much as I should be practicing. The month-long practice exercise imposed to me to realise my characteristic incompetence (Dahlstedt, 2012) in a live setting, but also to provide me with hope as I did make some progress. I certainly feel more confident after a month of daily practice, and I also believe I have developed a more on-the-fly feeling about the generated sound patterns. A live coder can quickly get into the labyrinth of 'coding for the sake of coding' and sometimes prioritising the code instead of the generated music could be the case. Because of the nature of typing and textual languages, live coding is far from traditional music performance. Maybe the only sensorimotor relation between live coding and instrumental performance is that both are carried out using serial skilled actions (Palmer, 1997).

The non-listening condition demonstrated how cumbersome and sometimes annoying it can be to live code for the sake of coding. When there is no anticipation of the musical outcome, it can be hard to get motivated to continue a live coding session. Sound enhances our anticipatory reward system, especially when combined with visual correspondences. On the other hand, this lack or reduced levels of anticipation because of absence of sound, may lead to unexpected musical outcomes, which can appeal to the coder. It can give a sense of being outside of one's self. Thus, a non-listening condition can be used to generate novel and creative sound patterns. For instance, I can have an educated guess of how a music pattern will sound by simply writing the code expression. An essential skill set of sound synthesis techniques can enable the coder to surprise herself positively. Thus especially in the case of online live streaming, a no-listening condition can be used as a creative technique for music-making. Given that the audience cannot understand the acoustic environment of the performer, the audience can also be unable to perceive whether the coder is listening or not to the generated sounds. Can such audience-performer dynamics bring about any novel interactions? Or do they raise any artistic concerns? Both are hard to answer and go beyond the scope of this study.

The visual helper I used during the sessions showed a limited number of UGens and was not developed to a musical agent. Its impact on my practice was minimal because it did not require attentional resources. On the other hand, I found such a simple program to be an excellent approach to familiarize myself with the large collection of UGens available in SuperCollider. I did a preliminary investigation on the potential of such visual aid, and I can see the potential of transforming this to either an on-demand agent or to a disruptive agent (Attanayake et al., 2020). Such a system would require a natural language processing component to be trained on the code and generate novel code chunks, coupled to a machine listening and learning component capable to extract similarity measures between the running musical outcome and the simulated code evaluations. The git repository can provide the coding database as it contains 98 individual live coding sessions.

It is important to notice that machine listening and learning algorithms have become more accessible.

In the seminal practice session by Collins and Olofsson only a pitch follower UGen is used in a couple of live coding sessions. More studies will follow on that aspect, given the rapid advances in the field of machine listening and machine learning. Indicatively the FluCoMa environment (Tremblay, Roma, & Green, 2021) offers a rich library for applying such computational techniques and is compatible with a variety of programming environments, like MAX/MSP, PureData and SuperCollider. In my sessions I did use some machine listening UGens, but I did not use any machine learning algorithms. This is likely the case because of the blank slate approach, which would require much effort in order to adjust such algorithms to my sessions.

While this is an ongoing research in terms of live coding practice as a disciplined endeavour, several things become apparent. First and foremost when there are no informative visual cues during the non-listening condition it is almost impossible to live code. I found myself loosing the thread of coding and had no idea how the musical outcome may sound like. Further on, from the spectrum alone it can be sometimes difficult to perceive whether command executions have actually any impact on the generated musical outcome.

A month-long practice is an excellent approach to get better on live coding. I found myself improving on several aspects, from creating naming conventions that can be informative, to how to separate triggers and control signals from sound generators. Such programming practices do have an impact on the musical outcome, as I was able to control musical structures more fluidly. Finally, I did carry the practice sessions alone and at times I was feeling demotivated and I was repeating my practices over and over. This feeling of demotivation amplifies the importance of collaboration. Whether the collaboration is as simple as turn-taking sessions or collaborative group performance, it is important to interact with more live coders and exchange ideas and knowledge. I recommend live coders that aim to commit to month-long practice sessions to find a human collaborator.

6. Conclusions

In this study, I challenged myself to do a month-long daily practice in musical live coding. I aimed to examine the relations between auditory and visual percepts and improve my live coding skills. I discussed how listening to the sound can help the coder in progressive evaluations and reduce error-proneness, while non-listening to the sound may be used as a creative coding practice technique. Further, when no informative visual cues and no sound are available to the coder, practicing musical live coding looks like an impossible task as I usually lose the thread of programming. I also experimented with a simple GUI visual helper, which can be developed into a software agent, and the documentation of the study can be used as a dataset for such developments. Practicing live coding in a daily and disciplined fashion is not easy, and several times, I repeated myself in both my coding practices and reflective diaries. The practice of musical live coding can be easier when it is a group activity, and I recommend to live coders who are determined to commit to month-long daily sessions to find at least one more coder.

7. Acknowledgements

I warmly thank the PPIG community for the valuable feedback during the conference, and special thanks to Alan Blackwell for his detailed review of the draft version of the present study. Both were catalytic to the improvement of this study.

8. References

Attanayake, U., Swift, B., Gardner, H., & Sorensen, A. (2020). Disruption and creativity in live coding. In *2020 ieee symposium on visual languages and human-centric computing (vl/hcc)* (pp. 1–5).

Blackwell, A., & Green, T. (2003). Notational systems–the cognitive dimensions of notations framework. *HCI models, theories, and frameworks: toward an interdisciplinary science. Morgan Kaufmann*, *234*.

Blackwell, A. F., & Aaron, S. (2015). Craft practices of live coding language design. In *Proc. first international conference on live coding.*

Blackwell, A. F., & Collins, N. (2005). The programming language as a musical instrument. In *Ppig*

(p. 11).
Burland, K., & McLean, A. (2016). Understanding live coding events. *International Journal of Performance Arts and Digital Media*, *12*(2), 139–151.
Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2003). Live coding in laptop performance. *Organised sound*, *8*(3), 321–330.
Dahlstedt, P. (2012). Between material and ideas: A process-based spatial model of artistic creativity. In *Computers and creativity* (pp. 205–233). Springer.
Dal Rì, F. A., & Masu, R. (2022). Exploring musical form: Digital scores to support live coding practice. In *Nime 2022*.
Davidson, J. W. (1993). Visual perception of performance manner in the movements of solo musicians. *Psychology of music*, *21*(2), 103–113.
Gamboa, M. (2022). Conversations with myself: Sketching workshop experiences in design epistemology. In *Creativity and cognition* (pp. 71–82).
Hove, M. J., Fairhurst, M. T., Kotz, S. A., & Keller, P. E. (2013). Synchronizing with auditory and visual rhythms: an fmri assessment of modality differences and modality appropriateness. *Neuroimage*, *67*, 313–321.
Jensenius, A. R. (2007). Action-sound: Developing methods and tools to study music-related body movement.
Magnusson, T. (2015). Code scores in live coding practice. In *Proceedings of the international conference for technologies for music notation and representation, paris* (Vol. 5).
Nilson, C. (2007). Live coding practice. In *Proceedings of the 7th international conference on new interfaces for musical expression* (pp. 112–117).
Palmer, C. (1997). Music performance. *Annual review of psychology*, *48*(1), 115–138.
Roberts, C., & Wakefield, G. (2018). *Tensions and techniques in live coding performance.*
Sorensen, A. C., & Brown, A. R. (2007). aa-cell in practice: An approach to musical live coding. In *International computer music conference* (pp. 292–299).
Tremblay, P. A., Roma, G., & Green, O. (2021). Enabling programmatic data mining as musicking: The fluid corpus manipulation toolkit. *Computer Music Journal*, *45*(2), 9–23.

Paper V

Reproducible musical analysis of live coding performances using information retrieval: A case study on the Algorave 10th anniversary

G. Diapoulis and M. Carlé

Proceedings of the International Conference on Live Coding (ICLC).
Utrecht, Netherlands. (2023).

https://doi.org/10.5281/zenodo.7843813

Proceedings of the 7th International Conference on Live Coding (ICLC 2023)
April 19th - 23rd – Utrecht, Netherlands

Reproducible Musical Analysis of Live Coding Performances Using Information Retrieval: A Case Study on the Algorave 10th Anniversary

Georgios Diapoulis
Chalmers University of Technology, University of Gothenburg
georgios.diapoulis@chalmers.se

Martin Carlé
Ionian University, Hub of Art Laboratories
mc@aiguphonie.com

ABSTRACT

We present a reproducible music information retrieval (MIR) study on 133 performances from the 10th anniversary of Algorave. Our aim in this paper is to provide a reproducible framework for computational analysis of musical performances. Here, we present a tool for analysing acoustical characteristics and for visualizing the musical structure from performances of one algorave event. Our musical analysis of the live coding performances highlights the musical diversity within the live coding community to a broader scientific audience. At the same time, we expect that the algoravers will gain insights on their own musical practices through the computational analysis of the musical structure of their performances. In concerning ourselves with reproducibility, our intention is to motivate more researchers to analyse musical practices of other under-represented music communities. As a basic tool for reproducibility we construct a pipeline for analysing performances using Python within a Jupyter notebook. To make this reproducible on different computers we wrapped the whole workflow setup into a docker image. We represent the results of our analysis as a series of plots of different kinds. These plots present both overviews of the entire repertory in compact form, and comparisons of individual pieces in more detail. In learning one can use such visualization as a means for raising awareness on one's evolution of the musical outcome. In performance this visualization can be developed to a real-time and possibly an interactive tool which informs the coder about the musical outcome of a live set on-the-fly. Finally, we reflect on how and to what extent such MIR studies can provide valuable insights in live coding performance practices, while also considering the limitations faced when dealing with such large parameter spaces in human machine musicianship.

1 Introduction

An algorave party aims at celebrating the act of live coding by hosting and promoting live music events across the world. Live coding is a performance practice where a domain specific programming language is used for music-making (Collins et al. 2003). Typically, live coders share their screen with the audience with the intention to make the process of music-making technically transparent to the listeners. Moreover, an algorave party is a political act in physically displacing academic music concerts from lecture halls to the dancefloors of nightclubs. During a live coding performance we allow ourselves to fail (Knotts 2020). Uncertainty and instability during a performance is common and the audience welcomes technical errors and programming mistakes during performance (Armitage 2018). These are attitudes also shared in other improvisation contexts.

Here, we present an analysis of recordings from the 24 hours stream of Algorave 10, retrieved from the Internet Archive[1]. The event was shortly announced only a few days before the happening by Alex McLean on the mailing list of TOPLAP[2] and the TOPLAP's chat server. The 144 available slots were filled within the next days and 133 recordings were retrieved from the archives. The aforementioned welcoming ethos of the live coding community was present vibrantly in the chat of the streaming provider and on other channels (e.g. TOPLAP Discord). All five continents were represented and a broad variety of performance practices and systems were exhibited during the 24 hours live streaming.

Our approach aims at a reproducible framework for music information retrieval of the Algorave 10th anniversary. The goal is to offer a framework that live coders can easily use to conduct musical analysis of performances and gain further insights into decoding a live set. We start by presenting the background of algorave parties and provide generic statistics of events taking place so far. We continue with the specifics of the Algorave 10th Birthday Party and offer descriptive statistics of the performances. Furthermore, we discuss our musical structure analysis from a cognitive MIR perspective and introduce a visual representation of the analysed musical forms. Finally, we review the structure of the reproducible framework of this study.

2 Background of algorave acts

The term algorave has been presented as a descriptor of a musical genre welcoming newcomers and underrepresented groups (Armitage 2018). Diversity and the live coding ethos are particularly important to the community. Accordingly, women live coders have been actively present since the founding act of TOPLAP ("TOPLAP ManifestoDraft," n.d.), albeit still forming a minority in the early days. Today, there are several bands, workshops, algorave events, TOPLAP nodes, and research programs (Sicchio 2014), where women are leading figures. For instance, the Hydra environment[3] for visual live coding, widely used in algoraves, has been developed by Olivia Jack.

The main impact of algoraves is to provide an incentive for people to get together and celebrate live coding dance music. As dancing involves the body ontological relations between concepts and sounds are further emphasised. In particular, drawing relationships between thoughts and sounds in electronic music is not a given thing (see algorithms in TOPLAP manifesto). There is a long discussion about whether electronic sounds can have an embodied meaning (Dahlstedt 2018), especially when the sound source has been disconnected from the musical instruments, transferred to electronic circuits, or reproduced through loudspeakers.

2.1 Historical accounts of algorave parties

An algorave is typically an event where people get together to enjoy live coding acts while dancing to the beat. The first algorave was in London in 2012 (Collins and McLean 2014). Since this kick-off event, many physical and virtual algoraves have been organised worldwide with the help of the algorave official website ("Algorave," n.d.). The 10th anniversary of the first algorave party was celebrated worldwide with numerous physical events and a 24 hours live stream. Each performance had a time slot of 10 minutes which is typically considered a short performance, just within the limits of being a challenging endeavour (Baalman 2020). This indicates how well the community has progressed since its beginnings in the early 2000s and suggests a grown maturity of the available live coding tools.

The Algorave page[4] lists 322 past events since the first algorave in 2012, and one future event that will accompany the ICLC 2023 conference this year. A crucial aspect of algoraves is that the events take place in a physical venue where people can get together and dance. A total of 21 out of the 322 events have been reported as being 'online' or on 'the internet'. The pandemic was a turning point for online events. Before, only two live-streamed sessions were reported. The first event was a hybrid algorave live-streamed from Sheffield in 2016 and the second was the 5th anniversary in 2017. The Algorave 10th anniversary live stream took place between 2022/03/19 and 2022/03/20, and was accompanied by several local events celebrating the occasion with the motto "weareten".

Collins and McLean (2014) presented the first study on algoraves. They show an estimate of how many people were dancing at each event out of a total of 18 algoraves, from 2012-03-17 to 2014-04-26. While it can be difficult to keep an objective account of how many people are dancing on algoraves, it is interesting to see the audiences' engagement with dance raising over the years. A way to approach this question would be to perform some video analysis on any recordings from algoraves and provide an estimate of the overall overt movement of the participants.

[1] https://archive.org/details/toplap
[2] https://forum.toplap.org/t/algorave-turns-10/1882/42
[3] https://github.com/hydra-synth
[4] https://algorave.com/ – Accessed 2022-12-12

2.2 Descriptive statistics on the 24 hours stream of the Algorave 10

The 10th anniversary had international coverage with 13 physical and virtual events. The events are mainly located in Europe and North America. Indicatively, in 2021 there were 15 live events in total, and in 2022 the total number of events is 28 so far. The motto "weareten" is explicitly written in several announcements. It is important to notice here that live coding in performing arts is about to become 20 years old, whereas the expression of live coding as dance in nightclubs is only 10 years old.

Throughout the 24 hours happening, the "Algorave 10th Birthday Party" was accompanied by a smooth and continuous live streaming of heterogeneous performances. Given the decentralized and flat organization of the community, we would like to draw special attention to the well-coordinated organization and execution of the event. Inevitably, different time slots served different time zones better. This led a clustering of locations within the 24 hours streaming. The event's coverage was worldwide and engaged musicians from all five continents (Africa, America, Australia, Asia, and Europe). Table 1 shows the number of performances per continent. There was one cross-continental performance between Barcelona and Toronto (slot #104) and one worldwide performance (slot #17)[5], where several live coders spontaneously entered Estuary[6] and started an on-the-spot improvisation.

EU	NA	SA	AS	OC	AF	Total
63	27	20	20	2	1	133

Table 1: The number of performances in Algorave 10 per continent. (EU: Europe, NA: North America, SA: South America, AS: Asia, OC: Oceania, AF: Africa).

Various programming languages and performance setups across the live coders can be noted. Table 2 accounts for the occurrences of the different live coding languages. The total sum of the noted systems equals the number of performances in the current analysis. Several performances used hybrid systems with more than one programming language. As annotated, we report the main language used during the performances. It is furthermore important that many systems also incorporated live coding visuals. In particular, Hydra was employed in many performances. Many video descriptions do not explicitly state the programming language used. Thus, Table 2 should be seen as the overall tendency of the performers' preferences for generating sound. Several performance setups also incorporated hardware synthesizers and other equipment. Most notably, a "game boy(girl)" by pulu, but also musical instruments, like electric guitars. Visual languages and any hardware controllers of musical instruments used during performances are not reported in our annotation.

LANGUAGE	Annotated
Tidal Cycles	52
SuperCollider	18
Sonic Pi	14
FoxDot	8
Orca	7
CSound	1
Chuck	1
MAX/MSP	1
Other	8
N/A	11
—————	————
TOTAL	121

Table 2: Annotated counts on the participants preferred music-making software programming environment.

[5] http://ten.algorave.com – Accessed 2022-05-10
[6] https://archive.org/details/algorave-10-equinox-open-jam

3 Characteristics of musical form

We begin by presenting theoretical backgrounds for the musical form of a performance from the perspective of music cognition. Then we discuss the parameters of the musical form and continue to build on this knowledge in order to construct a circular representation showing the temporal evolution of a music performance. Lastly, we discuss the benefits and limitations of this approach.

Musical form depends on our memory capacity. Musical memory operates on three main timescales, the so-called echoic memory for sounds with a duration of less than 0.5s, the short-term memory (STM) for sound events between 0.5-8s and the long-term memory (LTM) for sound events that occur on larger timescales, typically more than 15s (Snyder 2000). These different levels of organization surface differently when listening, imagining and playing music. For instance, during a music performance, the musician has to be attentive to both, the micro-structure of a musical phrase, and also to larger musical segments. In the context of a so-called canonical live coding performance (Roberts and Wakefield 2018), the coder is mostly applying changes to the musical outcome on larger timescales (15s and more). Few live coding environments may enable the musician to apply fast changes, like ixi-lang (Magnusson 2011) which was developed with the goal to make code modification within less than 5 seconds possible. A typical MIR architecture extracts low-level acoustical features. These features share some similarities with our auditory perception. The formation of pitch, for example, takes place within 10ms when listening to the acoustic environment. Above this level, our perceptual capacities group events together in order to structure perceptual boundaries (Bregman 1994). Indicatively, the gestalt principles, like the principle of proximity, similarity and good continuation are excellent examples of perceptual organisations (Schnupp, Nelken, and King 2011).

Several MIR architectures make use of the knowledge taken from the workings of our music cognition. For instance, computational segmentation of sounds is a typical example of a MIR task that employs music perception and cognition knowledge. Musical structure analysis uses it to identify different sounds, music excerpts or even songs. How different segments get organised is also important. Accordingly, pulse and meter are the first steps towards organizing motifs and themes. That is how we achieve periodic groupings of adjacent and nonadjacent elements (Parncutt 1994). Studying musical form requires to take both low-level and high-level music percepts into account. A typical cognitive approach of MIR begins with the low-level features to predict higher-level features. Consequently, a musical form can be described by two family characteristics: primary parameters, including pitch, rhythm and harmony, and a secondary parameters which include loudness, tempo, event density and timbre characteristics, among others (Snyder 2000). In this study we are combining these approaches. We start by extracting low-level acoustical features to synthesize higher-level musical structures and incorporate pitch-based and rhythm-based features to account for the musical form of a live coding performance.

3.1 Musical structure analysis in MIR

Musical structure analysis (MSA) in audio-based MIR focuses on objective descriptors of the signal. This is a necessary step when attempting to understand what the main acoustical characteristics of a composition are. Qualitative descriptors or human annotations typically complement these objective ones. The latter may correspond to information like labelling musical sections, such as verse, refrain, or annotating onsets and offsets of musical events, and so on. Furthermore, current state-of-the-art analyses are based on novelty, repetition and homogeneity (Klapuri, Paulus, and Müller 2010). Some of these approaches use similarity-based representations, such as MFCCs similarity matrices, to represent the temporal evolution of the audio. Studies on MSA have shown that the main challenges are related to subjectivity, ambiguity and hierarchy (Nieto et al. 2020). A typical example of "subjectivity" relates to how human annotations are collected. For instance, a noisy data set is a crowd-sourced annotation strategy. Respectively, ambiguity is related to the subjective percepts of the same annotator. Thus, when listening to a sound excerpt repeatedly, the annotator may be undecidable for a task such as music segmentation.

3.2 Primary and secondary parameters of musical form

3.2.1 Primary parameters

Primary parameters exhibit proportional relationships between them (Snyder 2000, 195). For instance, we do categorizations of pitch intervals, and we make subdivisions of rhythmic structures. The same does not apply to loudness, and pitch is independent of loudness. We perceive roughly the same pitch quality regardless of how loud a sound is. Pitch emerges when sound events range between 10-100ms and demonstrate stable perceptual qualities. For instance, we can easily understand whether a pitched sound has a higher pitch than another. Tonality and harmony are pitch-dependent, and when concerning music, we often discuss the fundamental note of a sequence. Sometimes an exact boundary between primary and secondary parameters is unclear. The mel-frequency scale is a pitched-based perceptual scale which describes how humans perceive equally distanced pitch intervals. A common acoustical descriptor is computed on the mel-scale. The so-called mel-frequency cepstral coefficients (MFCCs) have been used in many applications, like music genre classification, similarity measures, spectral envelope and speech recognition.

Rhythm is defined when "two or more events take place within the length of short-term memory" (Snyder 2000, 159). Rhythm emerges from repetition of sound events where pulse and event density are particularly important. In the context of musical structure, rhythm is expressed in hierarchical organizations and we learn and categorize different rhythms based on cultural characteristics, music training and musicianship.

3.2.2 Secondary parameters

Secondary parameters may be seen as by-products of primary characteristics. For instance, the spectral centroid, an acoustical feature which can describe the brightness of a sound, may be a by-product of pitched events. Brightness has a typical range of 2000-4000Hz which can be an important characteristic for periodic and transient signals. Pulse and event density are fundamental components of our rhythm perception. Pulse demonstrates an endogenous periodicity that is not strictly periodic but exhibits temporal fluctuations (Large and Snyder 2009). Beat depends on rhythmic complexity and has been suggested to play an important role in the perception of groove. Meter exhibits hierarchical organization and depends on the pulse, although there are cultural variances (Holzapfel 2015).

3.3 Musical form in live coding

Little research has been done on musical form and its characteristics in live coding. To some extent, this is reasonable as live coding is mainly an improvisational practice, free of rigid form characteristics. Recently a study examined two different aspects of visualisations during live coding practice (Dal Ri and Masu 2022), namely as a linear representation of events and as a density plot. The study suggests that the two approaches can complement each other. Magnusson examined how visual notation relates to code representation and how this is reflected in the temporal evolution of musical events. He also discussed how this knowledge was used for developing Threnoscope (Magnusson 2015).

4 Circular visual representation for STM windows

Given the above mentioned definition of rhythm focusing on our musical memory, we here present a circular visualization for structural representations of music performances. Our motivation is to represent performances of unequal lengths. Although all performances in this study had approximately a duration of 10 minutes, they all were still slightly different to each other. Therefore we present a circular representation to demonstrate the evolution of the Algorave 10 performances and we divide the circle into an arbitrary number of equally-distanced slices. Each slice has a maximum duration similar to the upper limit of our STM which is 8 seconds. The circular plot can represent both, pitch-related and rhythm-related characteristics combined. It may be used to represent the temporal evolution of one feature at a time. This is based on the assumption that each performance can be divided into an arbitrary amount of discrete segments. In this specific case, we divide the circular representation into 72 segments. As a result, the perceptual continuity between segments is imposed forcefully but it nonetheless facilitates our aim to communicate the temporal evolution of highly diversified performances. This is because several of the gestalt principles, like the principles of good continuation, common fate and similarity, are applied to the proposed visualization. The segments are equally-distanced from each other and their values are local descriptors which were calculated based on the mean feature values. Any other global descriptor can also be applicable, like higher-order statistical moments. Each segment is visually represented with a slice, as shown in Figure 1. An alternative approach to transforming all performances to equal lengths would need to involve dynamic time warping or zero padding. Dynamic time warping is a computationally expensive technique for our use case and doesn't serve well the reproducibility goals of the present study. Zero-padding could be useful for computations in the frequency domain. So, the circular representations here may be seen as an alternative approach to a frequency-domain representation, though they are not equivalent and should not be confused. Furthermore, we average real numbers per segment (feature values) and plot them in a circular representation to demonstrate their evolution over time.

In Figure 1 we map pitch- and rhythm-based features. Essentially, the plot is a counter-clockwise representation of the temporal evolution of a 10 minutes recording. The shift into the performer's perspective is done by representing the time as chunks (slices) of STM windows. The plot requires a three-dimensional representation. Time is the first dimension and two more dimensions show the feature values. Here, we show one pitch-based and one rhythm-based feature. Specifically, time is shown as a function of angles/radians (from $0\% - 100\%$), the pitch-based feature is shown on the colourmap (spectral centroid) and the rhythm-based feature is a function of the radius (onsets strength). The features are extracted using the Librosa library in Python (McFee et al. 2015).

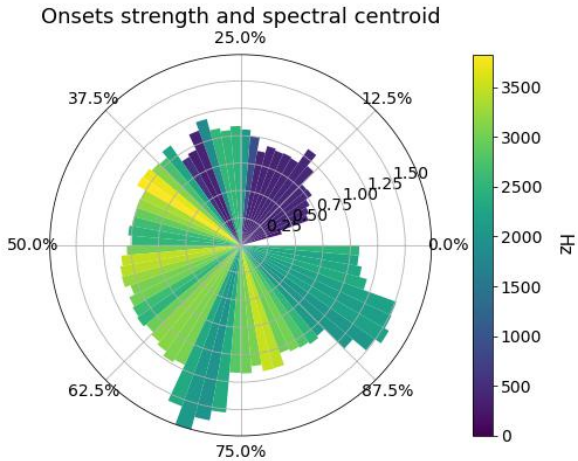


Figure 1: *Circular representation of a performance. The time axis is represented in radians, here shown in percentages. Each slice on the rose polar plot represents one STM window* $(0.5 - 8s)$*. The length of each slice shows the onsets strength, which may be seen as an indication of the density of the events. The heatmap shows the range of values for the spectral centroid.*

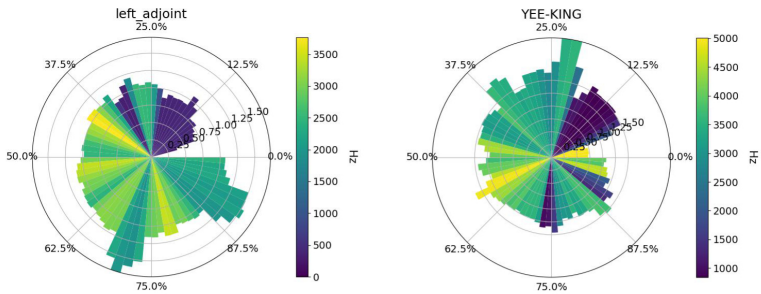


Figure 2: *A comparison of two performances, by left_adjoint and YEE-KING. The dimensions are the same as those in Figure 1. Please notice that the range of values for the acoustical features of onsets strenght and spectral centroid differ between the two performances.*

4.1 A closer look on two live sets from Algorave 10

Figure 2 shows two performances from Algorave 10, by left_adjoint[7] and YEE-KING[8]. The two cases were selected because they demonstrate a rich musical evolution over time. The plots show the same acoustical features as Figure 1, onsets strength on the radius and spectral centroid on the heatmap. Both performances begin with no sound during the first seconds of the performances. This is represented accurately with no sound events in the case of the performance by left_adjoint. In contrast, in YEE-KING's plot, some high-pitched background noise is adequate and is shown with high values of the spectral centroid, an indicator of the brightness of a sound and a relatively small density of sound events.

Both performances show a similar evolution in the first two minutes (see from 0% - 20%) with a low density of musical events and relatively low brightness values. During this period, left_adjoint conducts a so-called 'blank slate' live coding while experimenting with muffled synth and drum sounds. Later (~25%), some brighter drum sounds, like cymbals, are introduced, and the indicator of event density (onsets strength) is slightly increased. At around 40% of left_adjoint's performance, there is a climax in the brightness of the musical outcome caused by multiple cymbal sound events. Similarly, YEE-KING's performance demonstrates a climax on the onset strength in the region of 25% of the total performance time. In the second half of the performance (50% - 100%), both cases exhibit great amounts of variability

[7] https://archive.org/details/algorave-10-left-adjoint
[8] https://archive.org/details/algorave-10-yee-king

in the feature values. In the case of left_adjoint, two overt climaxes are shown of the event density indicator which is surrounded by variations between mid and high-frequency values for the spectral centroid. In the case of YEE-KING performance, the variations on the indicator of event density are subtle but there are significant variations on the spectral centroid.

5 Reproducible analysis of the Algorave 10 performances

Reproducibility with regard to both, information retrieval and analysis, has been realised by employing docker. In addition, the original videos were retrieved from the Internet Archive and placed in an online git repository for large files[9]. Since any changes on the git-lfs repository can be transparently tracked, the latter practice suits reproducibility better than retrieving the videos from the Internet Archive. However, a drawback of the git-lfs solution is that a configuration file with unique references to the original material is required. Another one it that repository provides put a limit on the download bandwidth which in our case means that after seven full downloads per month slower downloads are to be expected. We still consider this approach the better practice but remain open to any feedback from the community.

The `Dockerfile`, the recipe to build a container image, is made available in a GitHub repository[10]. The analysis presented in this study can be reproduced using two methods: i) run the prebuilt image from docker hub[11] which is the recommended method, or ii) build the image from the Dockerfile yourself[12]. There are three main steps to reproduce our results. The first step is to retrieve the recordings of the Algorave 10 performances from the mentioned git-lfs repository. We provide a Python script for the video-to-audio conversion. But the audio files are also readily available in mp3 format with as sampling rate of 22050 Hz. The second step extracts the acoustical features from the audio files, while the third step analyses and plots the acoustical features.

At the end of the building process of the docker image, the user can access a Jupyter Lab notebook demonstrating the descriptive statistics and three python scripts that render the plots of this study.

5.1 Acoustical feature extraction

The acoustical feature extraction is taking place during the building process of the docker image while the extracted features are stored in a data file. This file can then be accessed within the Jupyter Lab notebook.

For each performance we extracted 9 time series of acoustical features and one global descriptor for the estimated tempo: three rhythm-based features (onsets strength, tempo and pulse) and seven pitch-based features or feature vectors (20 MFCCs, spectral centroid, spectral roll-off, spectral flatness, spectral bandwidth, 7 features of spectral contrast and a pitch estimator of the fundamental frequency of a window). The pitch estimator showed many missing values ($\sim 40\%$ of the windows).

We mapped each feature to the circular representation as a function of time. Corresponding to the STM window of 8s, we computed a mean within each of 72 segments (as the actual low-level analysis window is 2048 samples). Finally we plotted each feature for 121 performances (as shown in Figure 3). Specific performances were excluded from the analysis as outliers. There were two main criteria for identifying outliers: i) performances which did not begin at the scheduled time, and ii) performances that showed low values of the onsets-related acoustical feature. The two main rhythm-based features, pulse and onsets strength (an indicator of event density) show similar distributions for every performance. Some pitch-based features, such as the spectral roll-off and the spectral bandwidth, show correlations. The two main rhythm-based features are also correlated with each other.

5.2 Estimated tempos of the algorave performances

Figure 4 show a histogram and a box plot of tempos for all the performances. The average tempo is 122 ± 18 bpm and is matching the so-called preferred tempo. The tempo range is $81 - 161$ bpm. The performances had an average duration of $574 \pm 42s$. This average duration results in an approximate value of $8s$ for each one of the 72 slices, which matches the upper limit of our STM.

[9] https://github.com/gewhere/algorave10-large-files
[10] https://github.com/gewhere/iclc2023
[11] https://hub.docker.com/r/algorave10/iclc2023
[12] Building the image from the Dockerfile require several hours. In contrast, the prebuild docker image runs within few seconds.

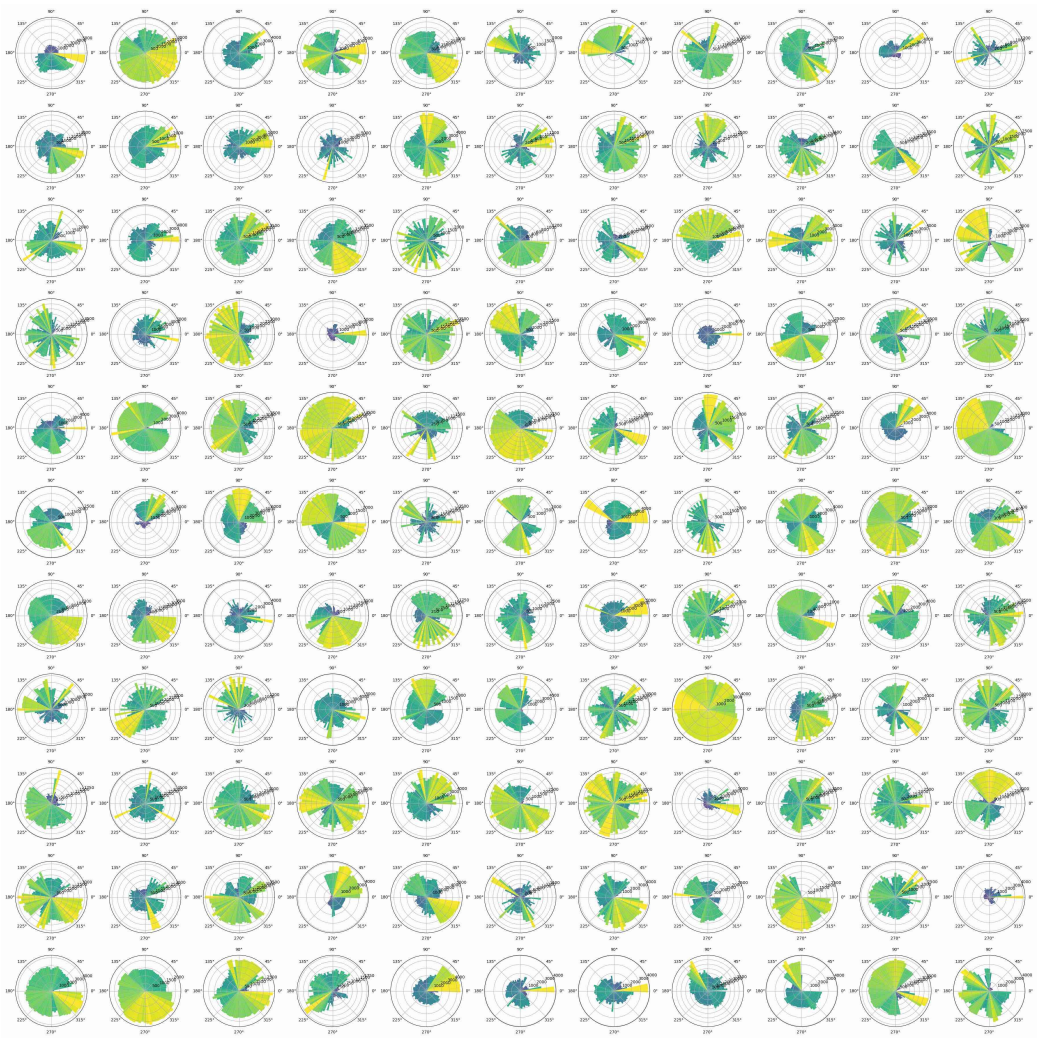


Figure 3: *Polar plots for all 121 live coding performances. The plots are two dimensional, in contrast to Figure 1, and show the temporal evolution of the spectral centroid. Every plot is normalized for each performance individually.*

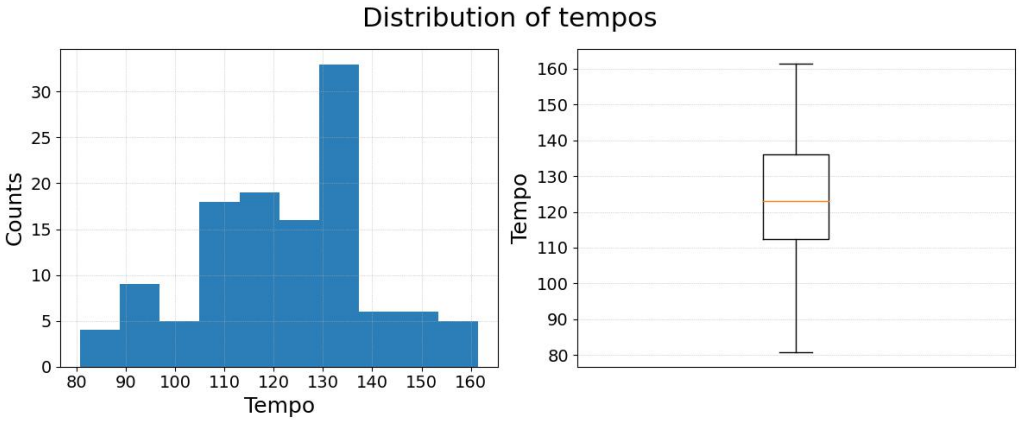


Figure 4: *Tempos for all the 121 performances. Left hand side shows a histogram of the tempos and the right hand side shows a box plot with the median and the 1st and 3rd quartiles.*

6 Discussion

The present study attempts to supply a transparent process for systematic computational analysis of live coding performances. As accessibility is an important value for the live coding community, we provide a reproducible analysis based on docker that launches a jupyter notebook for the computational analysis of the study. The raw data set which was retrieved from the Internet Archive has an audio-visual format from all the recordings carried out during the Algorave 10, 24 hours streaming. We copied the Internet Archive Algorave 10 collection to a git-lfs repository resulting in a transparent and computationally robust reproducibility method. As a beneficiary side-effect, this method offloads computational resources from the Internet Archive. Our study reveals first insights into the audio data set. To access further details of the live coding experience, one may look into computer vision analyses of the video material and how the outcome correlates with the acoustical features.

The live coding community acknowledges that there is no school for learning how to live code (Nilson 2007). Consequently, studying the musical structures of different performances can enable valuable comparisons of the musical forms created in live coding. To this end, the study offers a useful tool for studying the musical structure of live coding performances and provides a testbed for exploring more than the musical outcome. In particular so, as the video data are also made available, the basis for a multimodal information retrieval study is ready at hand. It is important for us to communicate our findings with the community and make the respective insights available to everyone interested in live coding practices. For this reason, we paid special attention to a quick and economic reproducibility of our study and offer a template to instantly start out with the MIR.

The produced diagrams allow for quick perception of similarity. For instance, the circular representation makes it easy to identify where a climax in the musical features occurs during a performance. In turn, this enables to visually recognise whether two performances share similar strategies in the evolution of the musical structure. We regard this as our main contribution and we also see potential for interactive applications. This circular visualization can convey valuable information of the evolution of cognitive processes and is used to communicate both, low-level and higher-level musical structures. For instance, the plots may be used to display perceptually significant changes on the level of our STM or to display larger musical segments, such as motifs and themes. To achieve this, it may be as easy as to highlight a quarter of the circular diagram. The capacity to easily spot out perceptually relevant information, such as climaxes, is an important feature, since a live coding session is a cognitively demanding task while resources are scarce during a performance.

Moreover, the proposed circular representation can be a valuable tool in teaching and practising live coding. For instance, it is acknowledged that cognitive load is high during a live coding performance (McLean 2014), but live coding is also seen as a technique which may sharpen attentional capabilities (Nilson 2007). While learning how to live code, the performer has to cope with many hard mental operations that may counteract the perceived flow during a performance (Nash and Blackwell 2014). Hence, an interactive visualization of the evolution of a live coding session may be useful, especially in the case when practising live coding. Such an interactive application could be seen as an informative real-time prop that reminds the coder about the stationary evolution of the acoustical features.

Besides the concrete case of circular diagrams, the study offers a playground for exploring similarities and differences between performances, live coding systems and languages. Further analyses including the video material can help to examine how programming language notations are related to musical structure. Such analyses can be valuable for making informed decisions in developing live coding idioms and languages.

Information retrieval studies in live coding are limited. Xambó, Lerch, and Freeman (2018) has proposed a theoretical framework for information retrieval studies in live coding focusing on real-time processes. Tools for such real-time applications have been available for over a decade (Collins 2011) and they overlap to some extent with technologies of musical agents (Xambó 2021). The approach we present here differs from such real-time and performative applications in focusing on an offline system for analyses of live coding musical performances. Such a reproducible testbed may help us to identify patterns during a live coding performance, ultimately leading to interactive tools for real-time applications. Indicatively, we envision an interactive application for our proposed circular visual representation of the musical form which might become a tool of descriptive notations (Magnusson 2015). More applications may come out of this engagement with a similar reproducible approach.

The diversity and the ethos of the community encourage open practices and transparent procedures. Likewise, accessibility has been addressed in live coding (Skuse 2020; Vetter 2020). Design and display technologies have been reported as key facilitators for disabled musicians. Whether the circular representation presented here may facilitate learning and interactive explorations in this direction is an open question. We would like to point out that we regard the circular representation as equally useful for both, disabled and non-disabled musicians. For example, in the case of visually impaired and blind musicians, we can imagine a tangible shape-changing interface for representations of the musical structure.

7 Conclusion

We present an information retrieval study of the Algorave 10, 24 hours live stream. A total of 133 performances with a maximum duration of 10 minutes each were retrieved from the Internet Archive and mirrored to a git repository, based on git for large files. We present a reproducible study using the docker containerisation infrastructure as to offer a playground for the creative exploration of information retrieval techniques applied to live coding performances. The computational analysis is based on audio data but the original recordings also include video data which can provide further possibilities for in-depth analyses of the performances. On the basis of our engagement with the dataset, we can provide descriptive statistics about the Algorave 10 and a circular visual representation for the musical form of live coding performances. This visual representation was constructed by considering aspects of music perception and memory. We expect that such visual representation will be useful for live coders and may have applications in teaching and practising musical live coding.

8 Acknowledgements

This contribution has been partially funded through the financial support of the project "ΔΗΜΙΟΥΡΓΙΚΟΣ ΚΟΜΒΟΣ ΤΕΧΝΩΝ *MIS :5047267*" code 80504, ΕΣΠΑ 2014-2020, ΕΠΑνΕΚ; HAL (Hub of Art Laboratories), co-financed by Greece and the European Union and implemented at the Ionian University, Corfu.

References

"Algorave." n.d. https://algorave.com.

Armitage, Joanne. 2018. "Spaces to Fail in: Negotiating Gender, Community and Technology in Algorave." *Dancecult: Journal of Electronic Dance Music Culture* 10 (1).

Baalman, Marije. 2020. "Marije Baalman performing 10 minute live coding challenge at Creative Coding Utrecht." https://marijebaalman.eu/projects/code-livecode-live.html. 2020.

Bregman, Albert S. 1994. *Auditory Scene Analysis: The Perceptual Organization of Sound.* MIT press.

Collins, Nick. 2011. "SCMIR: A SuperCollider Music Information Retrieval Library." In *ICMC*.

Collins, Nick, and Alex McLean. 2014. "Algorave: Live Performance of Algorithmic Electronic Dance Music." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 355–58.

Collins, Nick, Alex McLean, Julian Rohrhuber, and Adrian Ward. 2003. "Live Coding in Laptop Performance." *Organised Sound* 8 (3): 321–30.

Dahlstedt, Palle. 2018. “Action and Perception: Embodying Algorithms and the Extended Mind.” In *The Oxford Handbook of Algorithmic Music*, 41–66. Oxford University Press.

Dal Ri, Francesco Ardan, and Raul Masu. 2022. “Exploring Musical Form: Digital Scores to Support Live Coding Practice.” In *NIME 2022*. PubPub.

Holzapfel, André. 2015. “Relation Between Surface Rhythm and Rhythmic Modes in Turkish Makam Music.” *Journal of New Music Research* 44 (1): 25–38.

Klapuri, Anssi, Jouni Paulus, and Meinard Müller. 2010. “Audio-Based Music Structure Analysis.” In *ISMIR, in Proc. Of the Int. Society for Music Information Retrieval Conference*.

Knotts, Shelly. 2020. “Live Coding and Failure.” *The Aesthetics of Imperfection in Music and the Arts: Spontaneity, Flaws and the Unfinished*, 189.

Large, Edward W, and Joel S Snyder. 2009. “Pulse and Meter as Neural Resonance.” *Annals of the New York Academy of Sciences* 1169 (1): 46–57.

Magnusson, Thor. 2011. “The Ixi Lang: A Supercollider Parasite for Live Coding.” In *ICMC*.

———. 2015. “Code Scores in Live Coding Practice.” In *Proceedings of the International Conference for Technologies for Music Notation and Representation, Paris*. Vol. 5.

McFee, Brian, Colin Raffel, Dawen Liang, Daniel P Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. 2015. “Librosa: Audio and Music Signal Analysis in Python.” In *Proceedings of the 14th Python in Science Conference*, 8:18–25. Citeseer.

McLean, A. 2014. “Stress and Cognitive Load.” *Collaboration and Learning Through Live Coding*.

Nash, Chris, and Alan Blackwell. 2014. “Flow of Creative Interaction with Digital Music Notations.”

Nieto, Oriol, Gautham J Mysore, Cheng-i Wang, Jordan BL Smith, Jan Schlüter, Thomas Grill, and Brian McFee. 2020. “Audio-Based Music Structure Analysis: Current Trends, Open Challenges, and Applications.” *Transactions of the International Society for Music Information Retrieval* 3 (1).

Nilson, Click. 2007. “Live Coding Practice.” In *Proceedings of the 7th International Conference on New Interfaces for Musical Expression*, 112–17.

Parncutt, Richard. 1994. “A Perceptual Model of Pulse Salience and Metrical Accent in Musical Rhythms.” *Music Perception* 11 (4): 409–64.

Roberts, Charlie, and Graham Wakefield. 2018. “Tensions and Techniques in Live Coding Performance.”

Schnupp, Jan, Israel Nelken, and Andrew King. 2011. *Auditory Neuroscience: Making Sense of Sound*. MIT press.

Sicchio, Kate. 2014. “Hacking Choreography: Dance and Live Coding.” *Computer Music Journal* 38 (1): 31–39.

Skuse, Amble. 2020. “Disabled Approaches to Live Coding, Cripping the Code.” In *Proceedings of the International Conference on Live Coding*, 5:69–77.

Snyder, Bob. 2000. *Music and Memory: An Introduction*. MIT press.

“TOPLAP ManifestoDraft.” n.d. https://toplap.org/wiki/ManifestoDraft.

Vetter, Jens. 2020. “WELLE-a Web-Based Music Environment for the Blind.” In *Proceedings of the International Conference on New Interfaces for Musical Expression. Birmingham, United Kingdom*, 701–5.

Xambó, Anna. 2021. “Virtual Agents in Live Coding: A Short Review.” *arXiv Preprint arXiv:2106.14835*.

Xambó, Anna, Alexander Lerch, and Jason Freeman. 2018. “Music Information Retrieval in Live Coding: A Theoretical Framework.” *Computer Music Journal* 42 (4): 9–25.

Paper VI

Liveness and machine listening in musical live coding: A conceptual framework for designing agent-based systems

G. Diapoulis

Proceedings of the 4th Conference on AI Music Creativity (AIMC).
Brighton, UK. (2023).

ISBN: 9780995786295

AIMC 2023

Liveness and machine listening in musical live coding: A conceptual framework for designing agent-based systems

Georgios Diapoulis

URL: https://aimc2023.pubpub.org/pub/dk76kh2j

Abstract

Music-making with live coding is a challenging endeavour during a performance. Contrary to traditional music performances, a live coder can be uncertain about how the next code evaluation will sound. Interactive artificial intelligence (AI) offers numerous techniques for generating future outcomes. These can be implemented on both the level of the liveness of the code and also on the generated musical sounds. I first examine the structural characteristics of various live coding systems that use agent-based technologies and present a high-level diagrammatic representation. I sketch simple block diagrams that enable me to construct a conceptual framework for designing agent-based systems. My aim is to provide a practical framework to be used by practitioners. This study has two parts: i) a high-level diagrammatic representation informed by previous studies, where I analyze patterns of interaction in eight live coding systems, and ii) a conceptual framework for designing agent-based performance systems by combining both liveness and machine listening. I identify diverse patterns of interactivities between the written code and the generated music, and I draw attention to future perspectives. One code snippet for SuperCollider is provided and mapped to the conceptual framework. The vision of the study is to raise awareness on interactive AI systems within the community and potentially help newcomers navigating in the vast potential of live coding.

Introduction

Live coding is a performance practice where the performers share their screens with the audience and apply modifications to the running program[1]. It offers a rich technique for generative music[2] and a novel computing platform for exploring autonomy and interactivity with AI systems. Here, musical AI systems may range from rule-based systems to evolutionary computations, and from swallow learning to deep learning approaches. Liveness is here used as the “ability to modify a running program”[3] and is an inherent quality during a live coding performance[4]. The term liveness is not used similarly to its meaning in musical aesthetics[5] but rather denotes a degree when something is live[6]. I will mainly focus on a technical understanding of liveness as introduced by Tanimoto, which is mostly concerned about the live feedback to the programmer[7]. Machine listening is used during live performances for online sensing of musical percepts[8]. There is a long history of agent-based systems without support of real-time audio listening[9][10], but in the last decade, there has been immense progress in audio applications. Such developments have resulted in machine learning and machine listening ecosystems, like FluCoMa[11] and Sema[12]. Here, I will not focus on ecosystem designs but on performance systems developed by live coding practitioners. These are typically developed for a specific performance and are usually experimental designs.

Many practitioners can be unclear on implementing AI architectures into their designs. Whereas this may look unjustified due to decades of experience with interactive AI technologies[13], creative AI practices have a slow diffusion into interactive music systems. This slow diffusion is partly reflected in modern commercial software applications, which lean towards seamless interactions. The same issues apply to live coding systems to a

certain extent. Indicatively, there are recent efforts to implement machine listening in Chuck[1], and popular languages like Sonic Pi[2] and TidalCycles[3] do not offer built-in implementation for machine listening and machine learning. Thus, my goals are to raise awareness within the community and offer an entry-level for those unclear on how to incorporate interactive AI in their performance systems.

Here, I begin with a high-level diagrammatic representation of live coding and present a conceptual framework of agent-based systems in live coding. Musical agents are here seen as both human and software agents[14]. The framework is informed by examining various examples of live coding systems developed by practitioners and reflecting on my experience as a live coder. It addresses some affordances and temporal constraints to consider when designing interactions between code and music. It is an observational study where I reflect on my practice when necessary and provide one code example in SuperCollider. The example aims to help navigate the proposed conceptual framework when designing agent-based systems.

I will start with the related work on liveness and machine listening in live coding. Then I continue and present the methodology and the observation material. I examine the basic structural components of agent-based live coding systems and discuss interactivity patterns on the observed material. The last section presents a conceptual framework for designing agent-based systems in live coding.

Theoretical background

Tanimoto's hierarchy of liveness was initially presented with four levels[7]. The levels are informative (L1), informative and significant (L2), informative, significant and responsive (L3), informative, significant, responsive and live (L4). Later on, Tanimoto introduced two more levels, called tactically predictive (L5), and strategically predictive (L6). Liveness is an inherent property of live coding systems, and a system used in the performance is L4 liveness[15]. A promising area of investigation has been started looking at predictive models, which are moving towards L5 liveness. Recently, a few live coding systems have exhibited characteristics of tactical predictions[16][17][18]. Essentially, a tactically predictive system can inform the users of their programming behaviours, and a trivial case is the auto-completion mode of modern text editors. A strategically predictive system can perform intelligent predictions and examine the liveness concerning agency. Some sort of predictive modelling is typically used when we aim to advance from an L4 system to an L5 system, albeit not necessary. Here, it is important to notice that no system today claims to fulfil the requirements for making tactical predictions. There is also a critical view in the literature[19] on whether advanced levels of liveness, such as code previewing, can be useful to the performer, as the cognitive resources during a performance are scarce, and more information can be no more than a distraction[20].

Besides the technical characteristics, liveness in programming environments depends on the notation of the language and the environment itself[4]. In live coding, notation usually consists of the functional parts of code. However, secondary notation, like comments, indentation and syntax highlighting, can also play an important role in the dramatization of a performance[21]. In musical live coding, liveness can have more qualities as the

musical outcome and the humans involved are necessary parts. Thus the environment extends further than that of a typical programming environment and includes the notation (code), the musical outcome (music), and the musical agents involved (agents). Typically the performers, but some authors would argue for the importance of the audiences[22], and for a relational sense of ‘otherness’ that artificial agents can induce during a performance[23].

One functional characteristic of the human agent in a live coding session is that it can hear the musical outcome. As already mentioned, when the running program is rendered to musical outcome within a performance context, then the live coding system is necessarily within the L4 liveness[15]. The code generates musical outcomes sensed by the coder, who modifies the running program. Thus, there is a feedback loop between code and musical sounds as mediated by our auditory perception. The code is rendered to another modality (sound/musical outcome) and consequently to a continuous auditory stream (listeners' perception) that the coder can monitor.

Collins[24] presented a cookbook for machine listening in live coding. He presented two main categories of systems. For the first possibility, also known as “live coding control of machine listening”, there are two design decisions: (i) a feature-adaptive design which employs some feedback processes and (ii) an event-based design. For the feature-adaptive design, Collins implements a code with a pitch extraction algorithm that operates at 10ms. The output of the pitch extractor is within a feedback loop that controls the same sound generator. For the event-based design, an onset detector senses the environment with a microphone and triggers sound events. Regarding the second possibility, also known as “machine listening control of live coding”[24], Collins presented a timbral instruction set approach to let the machine listening component do the programming on this (imaginary) computer architecture. This system has a clock-based operation and follows a bottom-up approach to live coding as the levels of abstraction are progressively built on-the-fly. This system exemplifies how we may program a computer using another sensory modality. In this case, the machine listening component is a model of human hearing. The results of this analysis are applying progressive modifications to the instruction set, which can successively perform on-the-fly computations.

Introducing the observational material

Xambó[25] reviewed agent-based systems for musical live coding practices, providing the main material of my study. I focus on the systems that do afford both social interactivity and learnability. These are the two dimensions that Xambó introduced and denote whether the musical agents can cooperate (‘social interactivity’) and whether the system affords either on-the-fly or pre-trained learning (‘learnability’). I call these interactive AI systems, which can either learn from the environment or sense the environment. Thus, systems like Cibo[26] are excluded from my study.

List of video material

The methodology is formulated from complete observations, meaning that the observer did not interact with or influence the observed cases[27], and abductive inference[28]. I used as modes of investigation clues, metaphors, patterns and explanations as have been formed from personal practice as a live coder. I set certain criteria for selecting the observation material, a method known as criterion sampling[29]. The criteria I set for the observations are: i) there is an online video of the system, either performance, or a demo, along with a corresponding article, ii) the system is using the 'standard paradigm' for live coding, that is typing on a keyboard in a textual programming language, iii) the system affords learning or sensing, and iv) all examples used in Xambó's review[25] that fulfill the criteria (i), (ii) and (iii).

The selection of the use cases presented here is a combination of a search on Google scholar using the keywords "live coding" AND "machine listening", "live coding" AND "software agents", "live coding" AND "musical agents". The retrieved material was constrained to the first 100 entries for each query. Also, all entries from the International Conference on Live Coding (ICLC) repository on Zenodo[4] were retrieved, and regular expressions were used to search relevant articles. A total of 85 articles were retrieved from Zenodo. Some entries were excluded when the article was not written in English. Also, several systems were excluded because of mixed designs on the user interaction, such as modifying the system using both code and interactive graphical interfaces (e.g., interactive sonification, interfaces for education). The retrieved studies were last time updated on 2023-01-30.

The first selection criterion for a video demonstration implies a requirement that a practitioner has developed the system. The second criterion constrains the observation material to the arguably most common approach to live coding, that is, typing on a keyboard. The third criterion sets the requirement for interactive AI systems, and the fourth criterion is using bibliographical information from state-of-the-art live coding practices and agent-based systems. This resulted in a corpus of 8 videos, as shown in Table 1. I will briefly introduce all systems in the following bullet list, and in the next section, I will examine the systems in relation to one another.

1	Attanayake et al.[16]	https://vimeo.com/447733242
2	Autopia[30]	https://vimeo.com/349044280
3	Cacharpo[31]	https://vimeo.com/227332172
4	Flock[32]	https://vimeo.com/145109691
5	Mégra[33]	https://vimeo.com/321099751
6	MIRLCa[34]	https://youtu.be/ZRqNfgg1HU0

7	Ruler[35]	https://youtu.be/zjTL0DOCNBo
8	Wilson et al.[17]	https://youtu.be/2F1D8Harnkc

Table 1: List with the observational material.

1. Attanayake[16] and colleagues present a recommender system to suggest novel musical patterns to the user. The system uses Markov chains on the melodic patterns to suggest novel musical sequences. The system has three user modes: (i) continue writing code, (ii) execute the recommendation, or (iii) request a new pattern. An interview study was conducted where the participants indicated that the 'disruptive' mode was more enjoyable than the others. The system does not use machine listening.
2. Autopia[30] is a collaborative live coding system that uses audience voting and evolutionary computations. It uses genetic programming to generate prescriptive notation, in this case, code chunks that generate music which is neither ambiguous nor imprecise[36] and run the code independently. The system can perform on its own (without live coders) and also can be controlled by the coders.
3. Cacharpo[31] is a system that offers a co-performer for collaborative live coding sessions. The autonomous agent listens to the coder and extracts low-level acoustical features, which are progressively linked to higher-level features. Then, semantic descriptions are informed by machine listening, and a pre-trained neural network model maps acoustical characteristics to musical code patterns. The system has two modes; the coder either writes the code and the agent awaits, or the agent writes the code, and the user awaits.
4. Flock[32] is a collaborative live coding system that uses machine listening and a voting algorithm. The network dynamics of the system use evolutionary computations and incorporate a preference function informed by machine listening. Such preferences are visualized using descriptive notation[37], a representation of what is heard as the preference algorithm controls the final audio mix.
5. Mégra[33] is a system that uses probabilistic Markov models to generate new patterns on-the-fly. The system does not use machine listening. Instead, it is similar to Attayake's online training procedure, focusing on small datasets. Contrary to Attanayake, the system does not offer a code preview.
6. MIRCLa[34] is a music information retrieval (MIR) for querying audio samples using semantic tags from the cloud. The system is trained on a database of sounds and computes similarity measures and retrieves audio samples from Freesound ready to use in performance. It has three main functions: i) audio repurposing, ii) audio rewiring, and iii) audio remixing.
7. Ruler[38] is a rule-based system that explores generative spaces. The system was developed as an approach to on-the-fly programming a sound synthesizer. There is no machine listening implemented, the evaluation was conducted with human participants and similarity measures on the rule-based generator.
8. Wilson et al. [17] developed an agent that suggests code patterns in TidalCycles to the user. The model was trained on a large code database. Essentially, the system affords code previewing, similar one of the user modes used in Attanayake and colleagues.

Diagrammatic representation of the systems

From the above descriptions and the video observations, I identify four necessary parts of the systems: i) Human agent(s), ii) Software agent(s), iii) Code, and iv) Music. I abbreviate as Human (*H*), Software (*S*), Code (*C*), and Music (*M*) when necessary.

To facilitate understanding, I use block diagrams to represent the systems above using a rather simplistic representation of musical live coding (Figure 1). A typical live coding session can be schematically represented in the below diagram (*H*, *C*, *M*). The *H* writes text rendered to *C*, and *C* generates sound rendered to *M*. This is an incomplete representation, as the human also listens to the musical outcome during a performance (see transition labels on Figure 2).

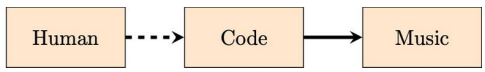


Figure 1: Simplistic schematic representation of a musical live coding session.

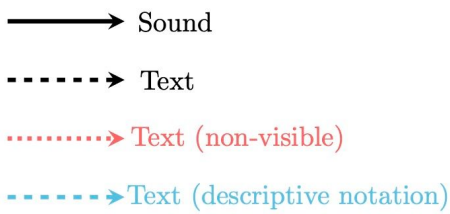


Figure 2: Descriptions of the transitions between the blocks.

Thus, a more accurate schematic representation of an agent-based system would include auditory feedback to the human (Figure 3). Here, the schematic representation does not show how the system informs the software agent. None of the systems in Table 1 can be represented in this manner, as they all have software agents aware of either the written code or the musical outcome.

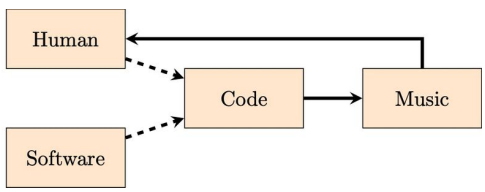


Figure 3: A simple schematic representation of an agent-based system for musical live coding, not corresponding to any of the observed cases.

More specifically, the systems by Attanayake et al.[16], Wilson et al.[17], and Autopia[30] are monitoring the written code and applying code modifications to prescriptive code chunks[37] (Figure 4). Some of the systems require the coder's permission to apply the changes, essentially a code preview feature that is not shown on the diagram below.

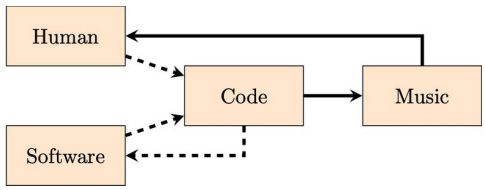


Figure 4: Agent-based systems informed by textual data.

Contrary to the systems above, Mégra and Ruler do not apply visible modifications to the code, as indicated by the red dotted arrow (Figure 5). Instead, the systems apply probabilistic and rule-based AI algorithms, respectively, to enrich the musical outcome as a seamless process. This functionality can be useful when fast musical variations are required, typically due to musical style requirements (e.g., dance music). Furthermore, Mégra has the potential for interactive visualizations, which can be seen as a descriptive notation. This feature is discussed by Reppel[33] but is not demonstrated in the accompanied video demo, and not visualized in the diagram below.

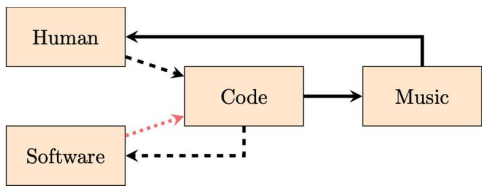


Figure 5: Agent-based systems informed by textual data without modifications on the prescriptive notation.

On the other hand, MIRCLa by Xambó[39], while agnostic about the content of the written code, the system is semantically aware of the musical outcome. The system has access to the acoustical features database on Freesound, and can retrieve similar context to the played sounds. This is denoted with the dashed arrow from *M* to *S*, indicating that the software agent is receiving textual data (Figure 6). Furthermore, while *S* is modifying *C* (red dotted arrow), this has no visible consequences for the user on the prescriptive notation. The user can only monitor the output of the interpreter to get informed about the applied code modifications. In practice, the system is much more complex, but I represent its main interactivity during a performance in the simple schematic.

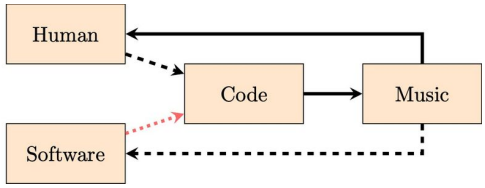


Figure 6: Agent-based systems informed by textual data related to the musical outcome, without modifications on the prescriptive notation.

Flock, by Knotts[32], is a system that applies a voting algorithm on collaborative live coding sessions. The analysis is performed on the musical outcome of the system, and the software agents modify descriptive notation, indicated with the blue dashed arrow, along with controlling the final audio mix (Figure 7). Again, the system is more complex as it is a multi-user performance.

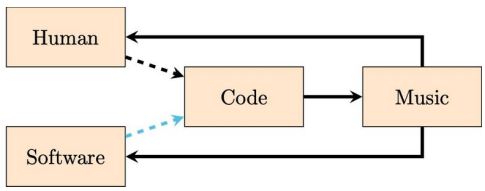


Figure 7: Agent-based systems informed by sound with modifications on the descriptive notation.

Cacharpo[31] is a system that simulates a co-performer and has a turn-taking design. The software agent awaits the user to provide permission to start typing novel code chunks. The system is performing an analysis of the acoustical features of the musical outcome and is not aware of the written code (Figure 8). Thus, Cacharpo generates prescriptive notation and listens to the musical outcome.

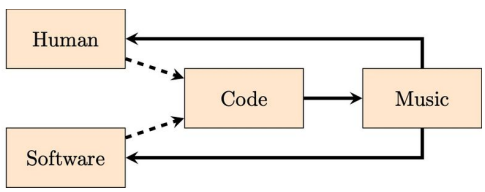


Figure 8: Agent-based systems informed by sound and applying modifications on the prescriptive code.

In all above cases the relations between the *H*, *C*, and *M* are always the same. The coder writes an encoded text which is decoded and rendered to music, and consequently, the coder listens to the sound to modify the code. From the observations, it becomes obvious that none of the systems is informed by both the music and the code

(Figure 9). More possibilities can be explored with this simple diagrammatic representation, and its complexity can increase when adding more components or transitions.

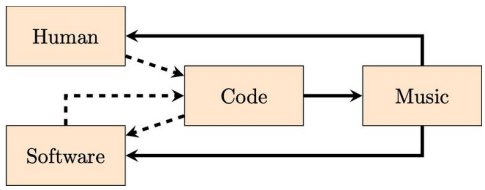


Figure 9: A future system informed by both the written code and the musical outcome.

Relations between the systems: Interactions of musical agents in live coding

Some systems incorporate software agents as assistants[31] or recommender systems[16][39]. There are several reasons for doing so, either because of intrinsic temporal constraints (e.g., typing speed) or because of extrinsic temporal constraints (e.g., browsing a large sample bank). For instance, Navarro and Ogborn[31] developed a software assistant (Cacharpo), which can be used for collaborative live coding performance between humans and software agents. The system uses machine listening and neural networks to generate novel code chunks in the autonomous agent workspace. This type of system can be seen as having intrinsic temporal constraints, as one performer cannot type the amount of code that two live coders can do. An example of extrinsic temporal constraints is MIRLCa by Xambó[39]. This system retrieves sound samples from the Freesound cloud database based on semantic queries (e.g., rain, noise). In this case, the live coder does not have the temporal capacity to search Freesound's webpage and select an audio sample that sounds like rain or noise. Consequently, MIRLCa system would do a better job given the constraints imposed on a live coder during a music performance, as it will calculate similarity measures between the semantic tag and the acoustical features of the sound sample.

On the other hand, Ruler, Mégra, and Autopia are agnostic of the acoustic characteristics of the musical outcome. These systems do not use machine listening. Instead, they apply AI algorithms in the domain of code. Whereas Autopia applies visible modifications on prescriptive code, Ruler and Mégra do not afford visible code modifications. The AI algorithms run on background processes, and, in the case of Ruler, the user is informed by the printouts from the interpreter. None of the three systems uses any descriptive notation, as Flock does, and none uses machine listening on the audio other than Flock and Cacharpo. To clarify, MIRLCa is applying machine listening-related technologies that use semantic information from the cloud. Thus, no real-time audio processing is performed, instead the system is informed from offline acoustical characteristics stored on the cloud.

All the abovementioned systems (Flock, Cacharpo, MIRLCa, Ruler, Mégra, Autopia) have L4 liveness. Below I continue with more cases which may be seen as advanced levels of liveness (L5 and L6). So far, two systems following the 'standard paradigm' have been developed towards making tactical predictions during a live coding session. These are the showcases by Wilson et al.[17] and Attanayake et al.[16]. The systems vary considerably in design decisions and available features. In Wilson et al.[17], the system is based on TidalCycles using a deep learning architecture pre-trained on a large corpus of code examples. In Attanayake et al., the system's predictive algorithm is a Markov model capable of online predictions of musical patterns. The two abovementioned cases make predictions on code segments, implementing a code preview functionality, with no involvement of machine listening. Their main difference is that Attanayake's et al. system affords online learning, whereas Wilson's et al. system affords offline learning.

It becomes obvious that prescriptive and visible notation is used by many systems (Autopia, Cacharpo, Attanayake et al., and Wilson et al.). The agents can adjust, or write from scratch, the prescriptive part of the notation, allowing the user to get involved with the generated code chunks. On the other hand, three out of eight systems (Mégra, MIRLCa and Ruler) do not use visible modifications on the prescriptive part of the notation or any descriptive notation. Only Flock provides the feature of modifications on the descriptive part of the notation. As for the modality of the music, it becomes evident that only Flock and Cacharpo use the acoustical characteristics for online sensing of the generated musical sounds, and MIRLCa applies offline sensing of acoustical characteristics using semantic information from the cloud. Similarly, some systems afford online training algorithms, and some afford pre-trained algorithms. I will further elaborate on these in the next section.

Conceptual framework for designing agent-based systems

As an extension of the previous section, where I examined the interactions of agent-based systems, I identified three different domains of interest when designing a system. The domains of i) 'Code', ii) 'Music' and iii) 'Software agents', will be referred to as the *coding domain*, the *musical domain*, and the *software agents*, respectively.

Below I introduce a conceptual framework (Figure 10) informed by technologies of liveness and machine listening. My focus is not on how to technically implement software agents in live coding but on what concepts can be useful when designing interactive AI systems. The framework is not meant to be exhaustive but rather a tool to aid in analysing and developing agent-based systems. Live coding is known to "resist or trouble any easy classification, categorization, or explanation"[40](p. 2). It is a non-linear framework that composes new knowledge from the literature and my musical practice. More specifically, Collins[24] recommendation on how machine listening and live coding may be combined is shown along with a part of a model for the timescales of auditory perception by Petri Toiviainen[41] on the musical domain. Some parts, like the temporal constraints and the affordances of the code, are formulated as a result of the literature study. The domain of software agents is constructed as part of the observation material. The thick semi-transparent line patterns indicate a

'weak' border between the three main domains of code, software agents, and music. Thus, the three domains span vertically (pattern coding), whereas the three categories of affordances, temporal constraints, and negative timescales span horizontally (colour coding). The outward arrows from the domain of *software agents* to the *musical domain* and the *coding domain* indicate that agents can act upon these modalities. As live coding also facilitates interactive visualizations, the framework constrains the creative aspects that can appear in a live coding session, as it does not consider the potential of visualization technologies. Finally, I provide a code snippet and map it on the framework to facilitate understanding of how to navigate the conceptual framework.

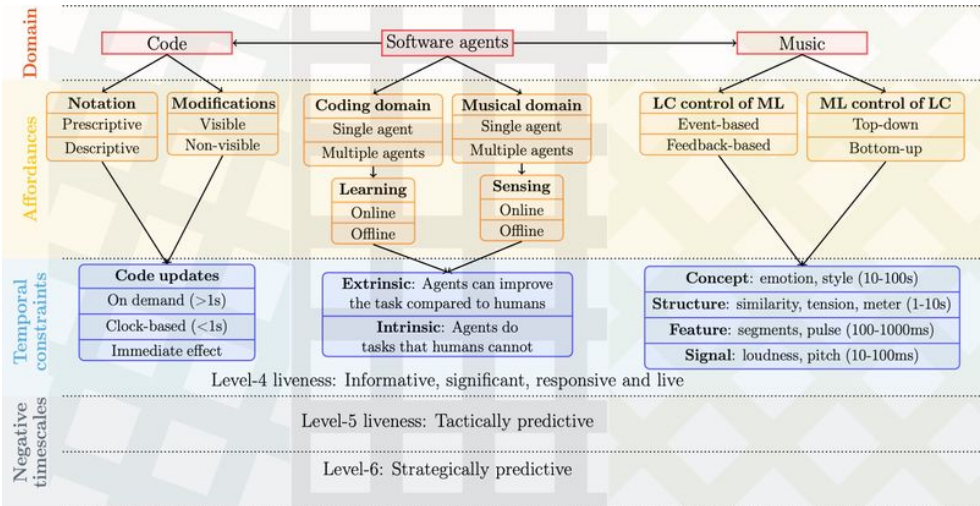


Figure 10: Conceptual framework for designing agent-based systems for musical live coding.

Domains of the conceptual framework

Coding domain

The leftmost column (Figure 10) shows the *coding domain*, along with some of the affordances of a system and the temporal constraints related to user interaction. The code can do an action (prescriptive notation) or can describe an action (descriptive notation). Secondary notation[22], such as code comments, is a descriptive notation. The visibility of the code is independent of whether the code is prescriptive or descriptive. Some processes can run in the background and be invisible to the user.

Moreover, the code updates demonstrate some inherent temporal constraints. The *coding domain* temporal constraints and affordances depend on actions that are either system actions, human agent actions, or software agent actions[42]. A question arises whether we consider the code part of the system, the coder's reasoning and performative processes, or an autonomous entity. Thus, it is related to how we ascribe agency to the code, and this opens a wider discussion on aesthetic appreciation[43], which go beyond the scope of the article. For

instance, Tidal has an inherent tempo clock which can be seen as a clock-based system action. Code updates can be performed by users or autonomous agents on demand or immediately (e.g., Attanayake et al.[16]).

Software agents domain

The middle column shows the domain of *software agents*. I identify that the agents can act upon the coding domain and the musical domain and can be either single agent systems or multiple agent systems. When they act on the musical domain they afford sensing, and when they act on the coding domain, they afford learning. Both sensing and learning can be either online or offline, as discussed above. I examined the relations between the different systems and I discussed how agents could exhibit either intrinsic or extrinsic temporal constraints.

Musical domain

The rightmost column show the *musical domain*. The affordances of the musical outcome are discussed as presented by Collins[24], and the temporal constraints present the temporal characteristics of low-level (signal), mid-level (feature) and high-level (structure, concept) acoustical features. There are indicative durations for each feature family, which indicate an inherent delay time for computations when applying machine listening.

Mapping use cases on the framework

Figure 11 shows examples of mapping different systems on the conceptual framework. At least one attribute from each block (e.g., Notation, Modifications, Learning, Sensing) is necessary unless a system is not operating on a specific domain. For instance, many systems do not incorporate machine listening technologies, like Attanayake et al.[16] which is agnostic of the musical domain.

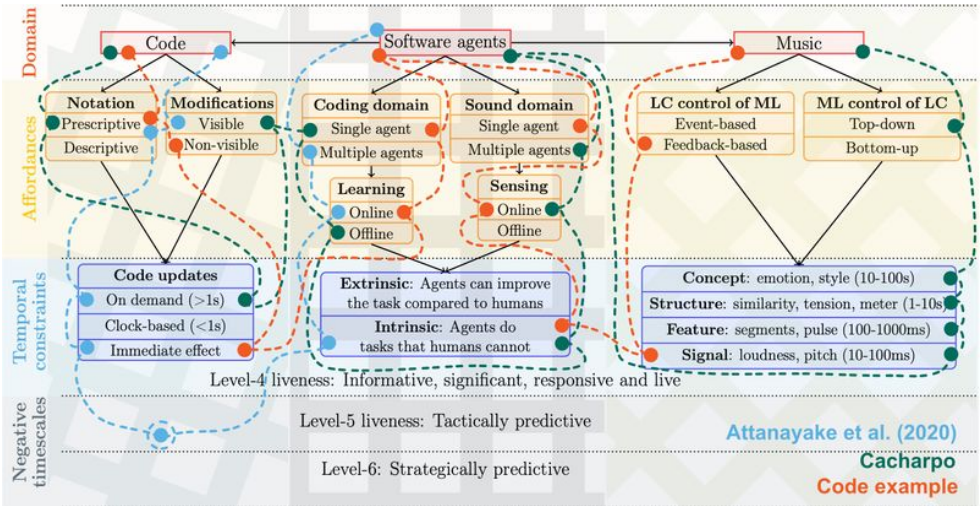


Figure 11: Mapping the code example and two of the systems on the conceptual framework.

The code example below demonstrates a simple agent-based system. If we follow the coding domain from top to bottom (orange dashed line on Figure 11), we can trace the path of interactivity starting from the musical domain (upper right on Figure 11). The code example applies feedback-based machine listening to extract a relatively low-level acoustical feature on-the-fly. The specific feature is the spectral centroid, a feature correlated to pitch and perceptual brightness. Thus, it affords online sensing using a single agent (musical domain). Following that, a conditional statement determines whether another single agent (coding domain) will generate sound or move into silence. This conditional statement (if statement in SynthDef 'agent') can be seen as a thermostat, which implements a negative-feedback design and can hardly be seen as a sufficient condition for learning by experience[44]. I will not attempt to discuss what can be seen as learning. Still, I would argue that this rather simple system exhibits some elementary properties that can be useful when designing systems that can learn to reach an equilibrium state. In complex systems, like a live coding performance, equilibria are not easy. Continuing in the coding domain, the code example, does not make any visible modifications on the notation and has immediate effect when reaching the threshold value.

```
// Code example in SuperCollider (SC3 v.3.13.0)

(
SynthDef(\ml, { arg audioBus = 0, controlBus = 2;
    var chain, feat;
    chain = FFT(LocalBuf(1024), InFeedback.ar(audioBus, 2));
    feat = SpecCentroid.kr(chain);
    Out.kr(controlBus,
            if(
                    RunningSum.kr(feat[0], 100) * 0.01 < 3000,
                    feat[0],
                    DC.kr(controlBus)
            ).poll(3)
    );
}).add;
```

```
arg freq = 330, ffreq = 2, pan = 0.0, amp = 1.0;
    var seq, pattern, trig, gate;
    pattern = [[1, 9/8, 2],     [3/2, 1, 0.5], [4/5, 3/2, 6/5]].mirror;
    gate =  Gate.kr(LFSaw.kr(ffreq).range(freq/2, freq),Impulse.kr(ffreq * 3));
    trig = Stepper.kr(Impulse.kr(ffreq), 0, 1, 9).fold(2, 7).fold(4, 5);
    seq = Demand.kr(Impulse.kr(trig), 0, Dseq(pattern * gate, inf));
    Out.ar(0, Pan2.ar(SinOsc.ar(seq).mean, pan, amp))
}).add;
SynthDef(\agent, {
    arg controlBus = 2, threshold = 3000;
    var seq = Stepper.kr(Impulse.kr(In.kr(controlBus)/100), 0, 1, 3);
    Out.ar(0,
        Pan2.ar(
            RLPF.ar(
                if(In.kr(controlBus) < threshold,
                    SinOsc.ar(In.kr(controlBus) * seq),
                        Silent.ar
                ), 3000, 0.5
            ), 0.0, 0.2
        ).tanh
    )
}).add;
)

// run line-by-line
~centroid = Synth(\ml);
~sine = Synth(\sine);
~agent = Synth(\agent);

~sine.set(\freq, 110);
~sine.set(\freq, 220);
~agent.set(\threshold, 100);  // deactivate
~agent.set(\threshold, 200);  // boundary
~sine.set(\freq, 1000);
~agent.set(\threshold, 1000);
~sine.set(\freq, 2000);
~agent.set(\threshold, 3000);
~sine.free;                   // free sine
~agent.set(\threshold, 100);  // autonomous
~agent.set(\threshold, 0);    // silence
~agent.free; ~centroid.free;
```

A similar logic can be applied to follow the traces of interactivities for Attanayake and Cacharpo, and in principle all use cases can be mapped to the framework. I would like to comment on the negative timescales, a term I adopted from Tanimoto's presentation[5] during the ICLC 2015. The dashed circle in Attanayake's system indicates that the system is not claimed to be an L5 system. I would claim that code previewing is L5 liveness, but it can be a complex issue whether code preview is L5 or not. Here, I support the authors' position (for both [16] and [17]) for not claiming to be L5 systems. As a last point on L5 systems, I would expect to see in the future liveness technologies that can compensate for machine listening inherent delays. Simply put, as the technical notion of liveness can 'see' into 'negative timescales' and machine listening has inherent delays due to digital signal processing constraints (e.g., sampling rate) and perceptual constraints, then I think that expectations are raised on these technologies[24].

Conclusions

In this study, I aimed to provide a practical framework for designing agent-based systems for live coding music performances. I reviewed studies focusing on the 'standard paradigm' to live coding, that is, typing on a keyboard, and I examined eight use cases with online video material. I presented a high-level diagrammatic

representation of live coding systems and identified interaction patterns between code, music and musical agents. I identified from this analysis that there is little attention to machine listening. Many instead incorporate machine learning for text generation which renders visible and prescriptive code chunks, but in many other cases, the agents have seamless consequences for user interaction. Also, no system incorporates both machine listening and text processing algorithms by making informed decisions on both domains of the generated music and the written code. Following the eight use cases analysis, I constructed a conceptual framework that can be used when designing agent-based systems and provided a code snippet to facilitate understanding. The implications of this framework are that it can help practitioners to navigate into ecosystems for machine musicianship, such as FluCoMa and Sema, and can also provide practical insights when designing agent-based systems for live coding, maybe to be used in education. Although the study presents a simple code example and two use cases of agent-based implementations mapped to the present framework, further informed decisions can be made using interviews with the developers of the use cases in question.

Ethics Statement

I declare no conflict of interest because of funding or otherwise. The present study follows the free/open-source software ethos along with low consumption of computational resources. No human participants were recruited for this study and no sensitive data were collected. The main methodology is based on an observational study from online video material where I did not interact with the participants, and it is mainly a methodological study. The study has an educational orientation and aims to support the live coding research community. Principles of accessibility, environmental sustainability, inclusion and socio-economic fairness were considered.

Acknowledgements

I thank Sara Ljungblad for pointing me out to connect and draw the trajectories of the interconnected components of the framework.

Footnotes

1. https://chuck.stanford.edu/release/VERSIONS — Accessed 23-01-28 ↩
2. https://sonic-pi.net/ — Accessed 2023-03-14 ↩
3. https://tidalcycles.org/ — Accessed 2023-03-14 ↩
4. https://zenodo.org/oai2d?verb=ListRecords\&set=user-iclc\&metadataPrefix=oai_dc ↩
5. Tanimoto’s keynote presentation during the first International Conference on Live Coding (ICLC 2015), https://youtu.be/4cJANuMiq18 ↩

References

- Attanayake, U., Swift, B., Gardner, H., & Sorensen, A. (2020). Disruption and creativity in live coding. *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 1–5. ↩
- Bernardo, F., Kiefer, C., & Magnusson, T. (2019). An AudioWorklet-based signal engine for a live coding language ecosystem. *Web Audio Conference (WAC 2019)*, 77–82. ↩
- Blackwell, A. F., & Collins, N. (2005). The Programming Language as a Musical Instrument. *PPIG*, 11. ↩
- Blackwell, A. F., Cocker, E., Cox, G., McLean, A., & Magnusson, T. (2022). *Live coding: a user's manual*. MIT Press. ↩
- Blackwell, A. F., Cocker, E., Cox, G., McLean, A., & Magnusson, T. (2022). Live Coding's Liveness(es). In *Live coding: a user's manual* (pp. 159–204). MIT Press. ↩
- Brown, A. R. (2016). Performing with the other: the relationship of musician and machine in live coding. *International Journal of Performance Arts and Digital Media*, *12*(2), 179–186. ↩
- Church, L., Nash, C., & Blackwell, A. F. (2010). Liveness in Notation Use: From Music to Programming. *PPIG*, 2. ↩
- Collins, N. (2003). Generative music and laptop performance. *Contemporary Music Review*, *22*(4), 67–79. ↩
- Collins, N. (2011). Live coding of consequence. *Leonardo*, *44*(3), 207–211. ↩
- Collins, N. (2011). SCMIR: A SuperCollider music information retrieval library. *ICMC*. ↩
- Collins, N. (2015). Live Coding and Machine Listening. *Proceedings of the International Conference on Live Coding*, 4–11. ↩
- Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2003). Live coding in laptop performance. *Organised Sound*, *8*(3), 321–330. ↩
- Croft, J. (2007). Theses on liveness. *Organised Sound*, *12*(1), 59–66. ↩
- Dahlstedt, P., & McBurney, P. (2006). Musical agents: toward computer-aided music composition using autonomous software agents. *Leonardo*, *39*(5), 469–470. ↩
- Diapoulis, G., Zannos, I., Tatar, K., & Dahlstedt, P. (2022). Bottom-up live coding: Analysis of continuous interactions towards predicting programming behaviours. *NIME 2022*. ↩
- Gifford, T., Knotts, S., McCormack, J., Kalonaris, S., Yee-King, M., & d'Inverno, M. (2018). Computational systems for music improvisation. *Digital Creativity*, *29*(1), 19–36. ↩
- Green, O., Tremblay, P. A., & Roma, G. (2019). Interdisciplinary Research as Musical Experimentation: A case study in musicianly approaches to sound corpora. *Electroacoustic Music Studies Network Conference: Electroacoustic Music: Is It Still a Form of Experimental Music?* ↩
- Knotts, S., & others. (2018). *Social Systems for Improvisation in Live Computer Music* [Phdthesis]. Durham University. ↩
- Lankoski, P., & Björk, S. (2015). Formal analysis of gameplay. In *Game research methods* (pp. 23–35). ↩
- Lewis, G. E. (2000). Too many notes: Computers, complexity and culture in" voyager". *Leonardo Music Journal*, 33–39. ↩

- Lorway, N., Powley, E., & Wilson, A. (2021). *Autopia: An AI collaborator for live networked computer music performance*. ↩
- Magnusson, T. (2019). *Sonic writing: technologies of material, symbolic, and signal inscriptions*. Bloomsbury Publishing USA. ↩
- Magnusson, T. (2019). *Sonic writing: technologies of material, symbolic, and signal inscriptions*. Bloomsbury Publishing USA. ↩
- McKechnie, L. E. F. (2008). Observational research. In L. M. Given (Ed.), *The SAGE Encyclopedia of QUALITATIVE RESEARCH METHODS* (pp. 573–576). SAGE. ↩
- McLean, A. (2014). Stress and Cognitive Load. In A. Blackwell, A. McLean, J. Noble, & J. Rohrhuber (Eds.), *Collaboration and learning through live coding* (pp. 145–146). ↩
- Nash, C., & Blackwell, A. F. (2012). Liveness and Flow in Notation Use. *NIME*. ↩
- Navarro, L., & Ogborn, D. (2017). Cacharpo: Co-performing Cumbia Sonidera with Deep Abstractions. *Proceedings of the International Conference on Live Coding*. ↩
- Olsson, C. M. (2015). *Fundamentals for writing research: a game-oriented perspective*. ↩
- Palys, T. (2008). Purposive sampling. In L. M. Given (Ed.), *The SAGE Encyclopedia of QUALITATIVE RESEARCH METHODS* (pp. 697–698). SAGE. ↩
- Paz Ortiz, A. I. (2022). *On-the-fly synthesizer programming with rule learning* [Phdthesis]. Universitat Politècnica de Catalunya. ↩
- Paz Ortiz, A. I. (2022). *On-the-fly synthesizer programming with rule learning*. ↩
- Reppel, N. (2020). The Mégra System-Small Data Music Composition and Live Coding Performance. *Proceedings of the 2020 International Conference on Live Coding*, 95–104. ↩
- Roberts, C., & Wakefield, G. (2018). *Tensions and Techniques in Live Coding Performance*. ↩
- Rowe, R. (1993). Chapter 5: Machine Listening. In *Interactive music systems: machine listening and composing*. MIT press. ↩
- Rowe, R. (2004). *Machine musicianship*. MIT press. ↩
- Stewart, J., & Lawson, S. (2019). Cibo: An autonomous tidalCyles performer. *Proceedings of the Fourth International Conference on Live Coding*, 353. ↩
- Tanimoto, S. L. (1990). VIVA: A visual language for image processing. *Journal of Visual Languages & Computing*, *1*(2), 127–139. ↩
- Tanimoto, S. L. (2013). A perspective on the evolution of live programming. *2013 1st International Workshop on Live Programming (LIVE)*, 31–34. ↩
- Toiviainen, P. (2015). *Lecture notes in Music Perception I*. University of Jyväskylä. ↩
- Wilson, E., Lawson, S., McLean, A., Stewart, J., & others. (2021). *Autonomous Creation of Musical Pattern from Types and Models in Live Coding*. ↩
- Wisdom, J. O. (1951). The hypothesis of cybernetics. *The British Journal for the Philosophy of Science*, *2*(5), 1–24. ↩
- Xambó, A. (2021). Virtual Agents in Live Coding: A Short Review. *arXiv Preprint arXiv:2106.14835*. ↩

- Xambó, A., Lerch, A., & Freeman, J. (2018). Music information retrieval in live coding: a theoretical framework. *Computer Music Journal*, *42*(4), 9–25. ↩
- Xambó, A., Roma, G., Roig, S., & Solaz, E. (2021). Live Coding with the Cloud and a Virtual Agent. *NIME 2021*. ↩

Paper VII

Musical live coding in relation to interactivity variations

G. Diapoulis

Organised Sound 28.2.
Cambridge University Press, UK. (2023).

https://doi.org/10.1017/S1355771823000444

Musical Live Coding in Relation to Interactivity Variations

GEORGIOS DIAPOULIS

Chalmers University of Technology, Gothenburg, Sweden
Email: georgios.diapoulis@chalmers.se

This article explores the similarities and differences between live coding and traditional music performances. The focus is on how bodily movements are expressed and whether pre-reflective processes may be activated during a live coding performance. While reports from practitioners vary on percepts of embodiment, the community is missing a theoretical background that reflects on practice. Understanding pre-reflective processes in live coding can benefit performance practices and tool development. As a live coder, I reflect on personal experiences and explore what I call 'interactivity variations', a term to denote different gestural manners of interactions during a performance. I observe patterns of embodiment among various live coders who use diverse performance systems from online videos. Out of 11 examples of performance systems, two cases demonstrate interactivity variations that can activate pre-reflective processes while another exploits direct manipulation. I present some implications for the patterns of bodily movement during a live coding performance and discuss how descriptive and prescriptive notation can be important and potentially influence our sensorimotor network. The article contributes a first account of a sensorimotor theory on live coding performances, reflecting on practice and embodied music cognition by presenting an aesthetic analysis of 11 online video examples.

1. INTRODUCTION

Live coding and traditional musical performance can be viewed as the two extremes on an imaginary continuum of music performance studies. While motoric and cognitive skills are demanding for both performance styles (Palmer 1997; McLean 2014), I will primarily concentrate on the qualitative differences that arise from the motoric skills performed through bodily gestures. This discussion will establish the foundation for my main research question: Are pre-reflective processes evident in live coding performances? The term 'pre-reflective process' is used interchangeably with 'fast processes' or 'subconscious processes' (Leman 2016).

My primary theoretical framework is drawn from music psychology and music perception research on music performance, employing Leman's theoretical framework on pre-reflective processes for expressive interactions. I subscribe to the phenomenological approach of autopoietic enactivism (Varela, Thompson and Rosch 2017), a sensorimotor theory of embodied cognition that sees cognition as an emergent phenomenon of sensorimotor activities. I also investigate how the user gesturally interacts with the input interface, which may involve addressing human–computer interaction (HCI) as needed. Essentially, the main emphasis is on the live coder, with a secondary focus on the physical interface and programming language. The field of research on music performance is largely indifferent to the specific musical instruments used. Instead, it focuses on musical structure, bodily movement and emotional responses (Palmer 1997). I concentrate on bodily movement and hope that future research will also look into musical structure and emotional responses, as they are currently unexplored areas in live coding.

Here, I describe how live coding is conducted by combining approaches from both HCI and music psychology, with an emphasis on gestural interactions. The term 'gestural interactions' refers to both musical gestures (Leman and Godøy 2010) and bodily gestures used in HCI, with gestures, viewed in the context of human–material interactions (Ishii, Lakatos, Bonanni and Labrune 2012), and the material, in this case, being sound. Musical gestures unify bodily movement and meaning and can therefore serve as a means of studying subjective percepts resulting from bodily movement (Jensenius, Wanderley, Godøy and Leman 2010). My motivation is to explore what can be learnt about a performer's cognitive and sensorimotor processes through observation of various systems and practices.

2. METHOD

In this article, I discuss 11 performance systems and practices, primarily by conducting complete and unstructured observations of live performances I have attended or watched online, reviewing a diverse corpus of literature and reflecting on my experiences as a live coder. Here, the literature spans music psychology and perception studies and research on live coding and HCI. Unstructured observations refer to the idea that an observation is carried out without systematic

Organised Sound **00**(00): 1–13
doi:10.1017/S1355771823000444

criteria guiding the observed showcases. It is an approach commonly adopted when there is little theoretical background on the phenomenon being observed or when the researcher does not know the outcome of the study (McKechnie 2008). Complete observations involve the observer refraining from interacting with the participants during the observation. In the case of live coding, complete observational studies have recently started to appear (Diapoulis and Dahlstedt 2021a) due to the increasing availability of online audiovisual material.[1] Ten years ago, such observations were not feasible due to the scarcity of online resources.

Thus, unlike observational studies in ethnomusicology, the methodology used here does not involve a coding scheme or other systematic criteria. Instead, the observation is an ongoing process documented through reflective diaries or other forms of documentation such as sketching diaries. Personal practices have certainly influenced the selection of the 11 video examples. Over the last year, I selected examples that I encountered, and together they formed a structured set of live coding systems that illustrate diverse practices of interactivity variations[2] on single-person examples (see section 5). Live coding is a community that documents itself (Haworth 2018) and reflects on itself, and my intention is to contribute to the research goals and ethos of the community, of which I consider myself a part.

3. TRADITIONAL AND LIVE CODING MUSIC PERFORMANCE

In traditional music performance, whether instrumental or vocal, there exists a direct relationship between energy and sound (Leman 2007). Put simply, the more effort we exert in striking a drum, the louder the resultant sound will be. This allows performers and audiences to engage with the process of sound generation, resulting in a more rewarding and enjoyable experience. Such a reward mechanism is related to our ability to make predictions between bodily movements and resultant sounds. For example, when observing a drummer lift her hand to the sky and strike the snare drum, we expect a loud and powerful sound as a result. Conversely, live coding does not involve this relationship (Dahlstedt 2018). In live coding, performers experience an indirect involvement with the music due to the use of notation (Nash 2012). There exists a dissociation between action and perception, as the bodily gestures performed by the live coder do not directly correspond to the production of sound. This can result in minimal bodily movement, which raises questions about the significance of typing on a keyboard (Haga 2008).

This indirect level of description is why live coding has been referred to as a *propositional improvisation* practice (Goldman 2019) and a radically different form of performance compared with traditional music performance (Sayer 2015). Goldman explicitly states that he does not practise live coding, while Sayer's study does not specifically reflect on practice. On the other hand, practitioners view live coding as necessitating a distinct range of sensorimotor skills. For example, Fredrik Olofsson does not feel any lack of immediate feedback, regardless of the non-physical interactions involved during coding (Nilson 2007).

Music performance is an embodied practice (Palmer 1997; Godøy 2021). Embodiment, here, 'assumes that subjective experiences are expressed in bodily changes' (Leman 2007: 236). During live coding, our subjective experiences include how we reason about the running program, how we appreciate the musical outcome and how we plan the progression of the performance (Diapoulis and Dahlstedt 2021b). All these are expressed in bodily changes, which can range from minimal keyboard movements to overt dancing to the beat. These movements can be synchronised and non-synchronised with the musical outcome.

Live coding practitioners have reported percepts of embodiment during performances, though such reports can be contradictory. According to Baalman (2015), embodiment is so profound in live coding that even grammars and programming languages can shape motoric execution patterns. By contrast, Hutchins (2015) sees a lack of embodiment in coding and aims to achieve it through live patching with synthesisers, because of tactility percepts. A recent workshop at NIME on gestures and embodiment in live coding also suggests diverse understandings of embodiment within the community (Salazar and Armitage 2018).

4. FROM NOTATION TO SOUND AND FROM SOUND TO MUSICAL MINDS

Live coding is sometimes described as a state of mind (Tanimoto 2015). It blurs musical concepts such as composition, performance and improvisation, while also challenging standard views on programming, such as the typical software engineering development cycle. The difference between live coding and most music improvisation practices is that live coding is an improvisation practice based on notation (Baily 1999; Magnusson 2019). Moreover, the notation is written on-the-fly, and can also be maintained with its accurate temporal evolution (Rohrhuber, de Campo,

[1]TOPLAP archive.org (https://archive.org/details/toplap), and YouTube Eulerroom (www.youtube.com/@Eulerroom).

[2]A term inspired by the established term in music psychology *performance variations*, which refers to different manners of expressive performance.

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

Wieser, Van Kampen, Ho and Hölzl 2007), thus can be reproduced by a machine. The standard paradigm of musical live coding involves a composer-programmer typing on a keyboard and sharing his/her screen with the audience. Roberts and Wakefield (2018) have referred to this standard paradigm as canonical live coding. This term refers to the practice that first appeared in the early 2000s, where a command line prompt was used as an interface equipped with an interpreter or a compiler. Interpreter-based languages are more popular due to their runtime immediacy on the musical outcome (Collins, McLean, Rohrhuber and Ward 2003).

When executable code chunks are rendered to audible sounds, the coder faces the indirect involvement between code and sound. Long sequences of typed individual characters are enfolded in the many steps required to evaluate a code expression. This is known as the complexity of the interface in HCI (Myers 1994), which refers to the number of steps required to perform a programmable action, such as opening a text document using drop-down menus on a text processor. In the cognitive dimensions of notation, a similar notion is that of the *closeness of mapping* (Blackwell and Collins 2005), a term from the psychology of programming that indicates how notation relates to output (McLean and Wiggins 2011). It is reasonable to assume that not all live coders are trained in blind typing techniques or use the same type of keyboard. Thus, the multiplicity of typing a single line of code can be enormous. Typing is also an open-loop motor program (Palmer 1997), which makes it an activity prone to errors, as there is no informative feedback mechanism to adjust or correct ongoing movements.

When listening to the generated sounds, the live coder somehow makes sense of the relation between the live writing of the code and the musical outcome. This may be seen as a mental link between the code and the sound, which enables the performer to listen to how structured code executions sound and imagine how novel code evaluations may sound. Thus, in live coding, both musical imagery and music listening have central roles during music-making (Diapoulis and Dahlstedt 2021a). Musical imagery is a mental process that can induce musical experiences or elicit musical realisations. It is a multimodal phenomenon, either auditory, visual or motoric, in which we anticipate desired effects or experience music both voluntarily and involuntarily (Jakubowski 2020).

Voluntary musical imagery, also known as *online musical imagery* to denote the imagery present during a performance, contributes to planning future actions (Bishop, Bailes and Dean 2012). Involuntary musical imagery can be induced by a phenomenon known as *notational audiation* (Brodsky, Kessler, Rubinstein, Ginsborg and Henik 2008; Keller 2012), a type of visual imagery. This is when we can internally 'hear' a melody depicted on a score. Motoric and auditory imageries are interrelated, and one can influence the other, also known as crossmodal interactions. A delay between performed actions and auditory feedback, which can be significant in live coding due to the complexity of the interface and the very poor closeness of mapping (Blackwell and Collins 2005), disrupts action-perception couplings, in turn disconnecting the temporal precision of motoric actions with listening percepts. This phenomenon is known as *delayed auditory feedback*, where action and sound are several milliseconds apart or *altered auditory feedback* when the performed action does not match the heard sound (Palmer 2012). These discrepancies can increase the gap between action and perception and hinder voluntary imagery and, consequently, embodiment. In this context, the visual aspects of the code or coding environment that can induce involuntary imagery (i.e., notational audiation) may play an important role.

When there are the disruptions in auditory feedback, how can we still embody the sound in our performance? The overt embodiment in live coding can be induced through listening to the sound, as we can exclude any tactility percepts that carry vibrations. Additionally, simulating motor actions through visual perception is possible, but it may be challenging for live coders to direct their attention away from the coding interface. In such situations, peripheral vision could be useful; for example, to be aware of people dancing nearby. Sometimes, we embody the generated sound by nodding to the beat or dancing. These are secondary aspects of musical gestures, which can have sound-accompanying or communicative functions (Leman and Godøy 2010; Jensenius et al. 2010) and do not have sound-producing functionality. Of course, this is more likely to happen when the musical outcome affords such embodied percepts, typically observed during Algorave parties, which are live coding events in dance clubs (Collins and McLean 2014).

Ultimately, the generated musical outcome facilitates an understanding and aesthetic enjoyment of the running program. One can say that the live coder uses the generated sound patterns as a proxy to form a mental model of the running program (Diapoulis and Dahlstedt 2021b). A mental model is 'any internal representation of the relations between a set of elements' (American Psychological Association n.d.) and denotes internal representations of relations and reasoning (Kosslyn 1996). Coding is an act of reasoning, although I will later argue that in live coding, reasoning on-the-fly can have more intuitive qualities when coupled with listening to the musical outcome, thus bringing forth percepts and imageries.

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

I build upon a sensorimotor theory of embodied cognition and adhere to the autopoietic enactive view of embodied cognition, which avoids distinguishing between mental and non-mental processes (Shapiro and Spaulding 2021). Here, I explain how we live code and how sound, mediated by code, affects our bodily experiences. As such, while the term *mental model* may not be the most historically accurate, I use it to describe the phenomenon I experience as a live coder or how we make our understanding from code to sound and vice versa.

We have seen, so far, that canonical live coding involves a complex interface consisting of typing on a keyboard. This amplifies our indirect involvement between the code and generated sounds and the perceived alteration of the auditory feedback, one that is not bound to physical actions. Besides these obvious challenges, musical notation is known to give rise to the phenomenon of notational audiation, which can induce involuntary imagery in performers. Musical imagery is known to give rise to percepts of embodiment and can also initiate motor activity. As we cannot engage with musical sound using sound-producing musical gestures or primary aspects of musical gestures, there is a dissociation between action and perception. Instead, we can only exploit secondary aspects of musical gestures, such as full-body movements, and examine how micro-movements such as typing can be useful. So far, I have only presented typing in live coding mostly as an activity that hinders embodied percepts based on action-perception theory.

4.1. How the interaction is performed

During a canonical live coding performance, the user interacts in a similar way to how end-users type in a word processor software. One persona of Nick Collins, known as Click Nilson, used a cloud-based[3] word processor to collaboratively re-write his presentation script with the audience during the live code festival symposium in Karlsruhe in 2013 (Nilson 2016). The main difference when a text editor is used alongside a programming environment for music-making is that the code evaluations are performed when a code chunk can be executed, which usually requires the conscious allocation of resources. Short edit-execution cycles, such as numerical parameter adjustments, are unlikely to enter conscious awareness, as this may take up to a few milliseconds. An average typing speed is 60–80 words per minute (wpm), or 3–4 characters per second (Marklin and Simoneau 2004). Short edits may be performed even faster than that, potentially approaching the upper limits of delayed auditory feedback, as the fastest typists can exceed several hundred wpm.

This strategy, which is predominantly found in full slate[4] live coding – an approach where pre-written code is used during a performance – is widely used by coders as a risk management technique or as a means of maintaining pace and flow (Roberts and Wakefield 2018). This full slate approach, a weak approach to live coding (Magnusson 2014b), exploits pre-written code. Thus, it can be common that the performer is only editing the code instead of writing the performance script on-the-fly, which Magnusson refers to as the strong criterion. Identifying the limit between a weak approach to live coding and simply a generative reproduction of the material can be difficult, similar to how hard it is to draw the line between interpretation and improvisation (McPherson and Limb 2019).

It is generally accepted that typing on a keyboard is an activity based on serial actions. This has similarities with traditional music performance, as *serial skilled actions* are carried out during a musical performance (Palmer 1997). (In fact, sequence production spans a variety of daily activities such as speech and walking.) The main difference between live coding and traditional music performances is that the live coder does not make sound-producing bodily motions. This makes live coding a performance practice that can hinder engagement with audiences and co-performers related to embodied experience and the aesthetic enjoyment of the music (Brattico, Brattico and Jacobsen 2009). The main reason is that when humans observe sound-producing movements, they can mentally simulate these actions in their sensorimotor network (Keller and Appel, 2010; Keller 2012). Sometimes this can also produce overt bodily movement, such as dancing (Keller 2008). The same does not apply to live coding, as the performed typing actions are mostly rendered in an unordered manner, or non-linearly, in terms of both temporal order and motoric sequential patterns. Commenting on the temporal aspects of live coding, Emmerson (2017: 115) states that 'the code writing of deferred time computer programming may be assembled out of time order'. In terms of how motoric sequences unfold, this means that the same typing sequence can be produced in different ways. Thus, audience engagement via mental simulations of observed bodily movements cannot easily happen. Instead, the audiences and co-performers tend to bodily entrain through music listening and bodily movement. Maybe the only aspect we can mentally simulate is an expectation about a new code chunk evaluation by simply seeing the visual highlighting of the code. Hernani Villaseñor (2019) has been performing live while filming his typing. This can be seen as a 'virtual action-sound mapping' (Jensenius

[3]I personally attended Click's Nilson presentation.

[4]Full Slate https://youtu.be/zJFJEqblEIA (accessed 14 April 2023).

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

2022: 155), denoting the multiple layers between action and sound, coupled with the discrete action of running the code. Live coders exploit this relation between code executions and resultant sounds. I usually do not look at the interpreter's output following every code evaluation; instead, I expect to hear the sonic changes.

During a live coding performance, novel code evaluations can have entirely predictable or not-at-all-predictable musical outcomes. Typically, a novel musical outcome may induce surprise to both performer and audience, expressed as a violation of expectations and may arise from creative explorations of the generative space (Dahlstedt 2012). Maybe the most common example of when our expectations are violated in a live coding performance is when a programming error occurs, and the generated sound patterns stop. The same can happen when we explore a generative music space, and the musical parameters suddenly tune into silence. This can make a live coding performance alien to an unfamiliar audience. The unfamiliar spectators may be ignorant of how the sound is produced, as reported in the documentary *Show Us Your Screens* (McCallum 2011). Maybe the audiences' perception will change as live coding becomes more popular. Magnusson (2019) reports that audiences unfamiliar with programming can follow the sound-generation process by attending to the live writing process. Visual aspects such as code highlighting are particularly important to this end, and the progress of special-purpose text editors, such as Gibber (Roberts and Kuchera-Morin 2012) and Strudel (Ross and McLean 2023) interfaces, has been tremendous in the last decade.

To summarise this subsection, canonical live coding involves typing on a keyboard. I have presented the activity of typing as serial skilled actions, which also applies to traditional music performance. However, whereas in traditional performance, the serial actions are tightly coupled in time and can give emergence to sensorimotor synchronisation, in live coding, the serial actions are out of time order. This suggests that using typing to interface with the programming language is not likely to result in the development of entrained bodily movements such as coordinated and rhythmic actions. Furthermore, it is unlikely that typing can induce mental simulations due to the multiplicities of individualistic typing styles, keyboard layouts and the error rate of open-loop motor programs. I can only speculate that exploiting typing gestures may induce mental simulations to co-performers and audiences on larger temporal chunks, where each micro-movement loses its significance and the gestalt of the typed chunks matters. But would that apply only to those exposed to live coding practice? Is familiarity important here? These are some open questions.

4.2. Pre-reflective processes in machine musicianship

The effect of action and perception on music cognition has been studied extensively. There is growing evidence that these two are inseparable from one another. We make our understanding by enacting our world (Varela et al. 2017), and the role of the human body is of fundamental importance. Here, musical gestures are seen as non-verbal communication, which has pro-social functions and can enable 'pre-reflective experiences' (Leman 2016: 92). For example, in electronic dance parties, a sweep from low to high frequencies may result in excitement, manifested in people raising their hands. How pre-reflective (or subconscious) processes are manifested in live coding has been barely discussed (Diapoulis and Zannos 2014; Dahlstedt 2018). This can be important as it will enable a discussion on how to better conduct live coding sessions and, consequently, how to better coordinate live coding ensembles.

I have already discussed that traditional music performance is tightly connected to bodily movement, but the movement is also related to expressive performance. Several mechanisms facilitate musicians' engagement with expressive performance (e.g., planning and anticipating future actions), entrained processes (e.g., loosely coupled bodily movement) and reward mechanisms linked to emotional resonance and aesthetic enjoyment (Leman 2016). As the embodied paradigm of music performance has gained more attention, entrainment is considered one of the fundamental mechanisms during music performance (Repp and Su 2013; Clayton et al. 2020). Entrainment is a phenomenon that appears in several mechanical systems and may bring about stability in oscillatory systems, regardless of any small perturbations within a system. Recent studies suggest that sensorimotor synchronisation (SMS) and coordination are responsible for the emergence of entrained processes during a duet performance (Clayton et al. 2020). SMS is a pre-reflective process roughly within 200–2000ms (ibid.), whereas coordination emerges at larger timescales of more than two seconds but may indicate coordination over several seconds to minutes. Godøy (2021: 2) also identifies roughly the same time intervals (0.3–5s) as being of utmost importance during a performance, describing it as the 'enigmatic relationships between notions of continuity and discontinuity in both philosophy and psychology'. Indeed, timescales below 200–300ms are mostly concerned with sound quality, including its early reflections within the acoustic surrounding. For longer durations, we mediate sound through our bodies and make music realisations. Over larger timescales, we coordinate, and social phenomena emerge as the outcome of complex interactions, such as a music performance in public.

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

These indicative timeframes would suggest that SMS is essentially impossible during canonical live coding. Sayer (2015) supports this argument while Goldman (2019) suggests that live coding can be seen as a *propositional improvisation* practice, which operates on long-term memory mechanisms. Goldman (2019) and Sayer (2015) suggest that live coding incorporates slow, conscious processes, making the emergence of fast, pre-reflective experiences impossible. Several reasons are offered for this, such as the content of feedback, the temporal nature of the feedback and the discrete nature of decision-making (Goldman 2019).

It is indeed true that the so-called canonical live coding incorporates mostly slow processes. Several systems, such as ixi-lang and TidalCycles (McLean and Wiggins 2010b), have been developed, taking such perspectives into account. Indicatively, ixi-lang was developed with the constraint of writing and executing a code chunk in less than five seconds (Magnusson 2011). Even this may not help in enabling fast, pre-reflective processes in a live coding session. So, the question is immanent: Can pre-reflective processes be enabled during live coding performance?

4.3. Summary on interaction, perception and cognition

I would argue that from a theoretical perspective, fast and entrained processes are unlikely to occur when using typing-based live coding systems. Entrained processes can afford small perturbations and eventually reach stability, but the multiplicities of typing a single line of code can be devastating to SMS. We can see bodily entrainment in live coders primarily as dancing to the beat, such as in the case of Algorave parties. But these are merely secondary aspects of musical gestures. It is unclear how these can influence the production of micro-movements. Maybe systems that require short typed sequences to make a sound (e.g., ixi-lang, TidalCycles), albeit being out of order and non-synchronised, can provide the best chance for the case of canonical live coding to employ pre-reflective processes, along with practices such as short edits. Orca,[5] a two-dimensional control interface inspired by trackers that affords single letter commands (for a description, see Blackwell, Cocker, Cox, McLean and Magnusson 2022: 147), operates on stream-based evaluations, which helps to reduce the delayed auditory feedback.

5. LIVE CODING HERE AND THERE

In this section, I present a selection of live coding examples. The reasons are twofold: first, to elaborate on how non-verbal interactions, rendered as musical gestures, are experienced during a live coding music performance; second, to discuss what can be learnt by observing such gestural interactions about the live coders' sensorimotor control and, possibly, cognitive processes. I say possibly because we typically conduct sophisticated experimental designs to merely infer what a person is imagining (Shepard and Metzler 1971; Zatorre and Halpern 2005), which is likely impossible using complete observations.

All the examples discussed here are solo performances, constraining the observed possibilities of interactivity variations. Furthermore, most video recordings from Eulerroom on YouTube or other individual resources, are solo performances. Hence, I limited my study to solos because the embodied interactions I study (e.g., serial skilled actions) are evident when a soloist is bodily interacting with the computer interface, and I do not study inter-musician interactions.

The online video examples were intended to be available to the general public (Snee 2013; Theodosopoulou Bourlogianni, 2021) and no consent was required (Loveday, Woy and Conway 2020) to conduct the observational study and the aesthetic analysis. The links to the videos are shown in Table 1. I compiled this specific list because each video is a good example of each particular system's characteristics. No computational analysis, download or reuse were conducted for this study.

The examples presented in the table are divided into three categories: 1) canonical systems, 2) bottom-up systems, and 3) a mixed category of systems. Canonical systems exemplify the standard paradigm to live coding, as discussed earlier. As canonical live coding is very well known in the community, I will present only one example of a canonical system which really stands out (Baalman 2009). Bottom-up systems are live coding systems that depend on very little abstraction (Diapoulis, Zannos, Tatar and Dahlstedt 2022). Typically, the levels of abstraction are built up on-the-fly, as we go. They are relatively uncommon and highly constrained, which exhibits some interesting characteristics. Also, the bottom-up systems in Table 1 do not use the keyboard, which makes them radically different from canonical systems. The third category presents systems that afford gestural input as direct manipulation (e.g., a mouse), an interface that affords continuous control and facilitates recognition instead of retrieval. I restricted the scope of this category and excluded non-conventional interfaces as the bottom-up category already uses a number of examples with unconventional interfaces. The third category of mixed systems demonstrates some examples that lie between canonical and bottom-up systems, as input control in bottom-up systems mostly relies on recognition instead of retrieval.

[5]https://100r.co/site/orca.html.

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

Table 1. Chronological list of performance systems in the observational study. *Betablocker*, *Code LiveCode Live*, *CodeKlavier CKalcuλator* and *iMac music* have earlier release dates than the release date of their corresponding videos.

Performer	Performance/System	Video URL	Category	Year
Dave Griffiths	Betablocker	https://vimeo.com/24390484	B	2006
Dave Griffiths	Al-jazzari	https://youtu.be/Uve4qStSJq4	B	2007
Marije Baalman	Code LiveCode Live	https://vimeo.com/434679284	C	2009
Georgios Diapoulis	stateLogic machine	https://vimeo.com/43121821	B	2012
Jonathan Reus	iMac music	https://vimeo.com/205714278	B	2012
Thor Magnusson	Threnoscope	https://vimeo.com/63335988	M	2013
Chris Kiefer	Approximate Programming	https://youtu.be/WwhpRtxq1Kg?t=3417	M	2015
The Duchess of Turing	Using various	https://youtu.be/hfJTF3KTnFM	M	2019
Noriega, F. I. and A. Veinberg	CodeKlavier CKalcuλator	https://youtu.be/hD-PWNDebD4	B	2019
uiae	Using PD	https://youtu.be/A-HohsA9R1o	M	2020
Fredrik Olofsson (redFrik)	SuperCollider	https://youtu.be/lqi-Vqr0qk4	M	2022

Note: B = bottom-up system; C = canonical system; M = mixed category of systems.

Before going to the examples, I want to clarify any confusion in the literature between low-level and bottom-up processes. As Roberts and Wakefield (2018) explain, low-level processes are mostly related to algorithms that operate on digital signal processing (DSP). These also afford low-level computational abstractions but do not necessarily generate or operate on formal languages. By contrast, bottom-up systems necessarily rely on formal languages. DSP algorithms are mostly engineering abstractions for time series analysis and computations. A time series is a sequence of data points, and some applications include forecasting future data points and smoothing noisy data, among others. As such, their primary function is to act as filters of information. A formal language or a computer architecture is not a filter; rather, it affords the universal process of computing any problem given enough time under Turing completeness (a Turing complete machine is a universal machine that can approximate any computable mathematical problem).

I did not include in Table 1 a variation of Betablocker (Bovermann and Griffiths 2014) written for SuperCollider,[6] as it would fit within the canonical live coding systems. In this case, Betablocker (Griffiths 2006) is used as a sound engine, which would spark a new category of low-level systems that can afford Turing completeness. Essentially, the system traverses from canonical live coding to bottom-up systems that generate low-level DSP processes. This example demonstrates how hard it can be to apply 'any easy classifications' to live coding systems (Blackwell et al. 2022: 231), as 'attempts to define this wide field ... are likely to become an exercise in herding cats' (Magnusson 2014a: 14).

[6]Tai-studio, An Introduction to Detablocker: https://vimeo.com/32938807 (accessed 14 April 2023).

5.1. The case of canonical live coding

Canonical live coding is the most common approach. This includes the 'standard' interface of a computer screen and a keyboard. Whether gestural actions can be performed on a keyboard is a matter of debate; however, the current consensus is that typing on a keyboard is 'neither observed nor significant' (Jensenius et al. 2010: 16). While this is undoubtedly the case in everyday interactions with computers, the same argument cannot hold in a live coding performance context. Typing is typically observed in this scenario as the performers share their screens. Furthermore, this becomes significant at certain times, specifically when code evaluations are performed and rendered as perceivable sound patterns. The sound vibrates molecular structures in the affine medium, that is, gas, liquid or solid. This transforms the meaning of typing into significant actions, as Baalman (2009; Baalman n.d.) demonstrate in the Code Live Code Live video (Table 1). Baalman switches on the onboard microphone of the laptop, and the typing sounds are used as raw material for the musical outcome. This performance setup demonstrates how typing on a keyboard can be both observed and significant.

Baalman (2015) has reported that our programming language of preference can influence our motoric execution patterns. She elaborates more on embodiment during a performance and provides anecdotal evidence of typing automaticity. The argument is that certain typing tasks have been automatised to such an extent that they require minimum effort. These are known as body schemas, cognitive organisations of one's body that can also monitor sequences of bodily

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

motions and appear as learned motor patterns (Leman and Godøy 2010). Whether gestural unfoldings, a term to indicate a weak temporality[7] of bodily gestures, may influence our mental model of the running program is an open question (Diapoulis and Dahlstedt 2021a). To elaborate, can this weak temporality of typing influence how we internally represent code-sound relations and reason about the running program? Godøy (2004: 58) has argued that we can mentally 'compress' bodily gestures in time using imagery, what he refers to as gestural imagery. The same cannot apply to sounds: we cannot mentally compress a sound and experience the same characteristics. Maybe gestural imagery can compensate on-the-fly due to our frustration with attending to every single moment of sound and actually influence our mental model (by encodings on our sensorimotor network). Baalman reports that our programming language of preference can shape our motoric actions so that certain sequences are encoded in our sensorimotor network as typing gestures that unfold in time when necessary. If the language of our preference can shape our motoric patterns, then can motoric patterns shape the language we use? Simply put, can the performer's gestures influence the development of the programming language? I will elaborate on these two questions in the next section.

5.2. The case of bottom-up live coding

I argue that bottom-up systems exhibit *interactivity variations* where the gestures can influence the programming language. I will mainly focus on two cases: CodeKlavier CKalcuλator by Noriega and Veinberg (2019) and the stateLogic machine (Diapoulis and Zannos 2012). These were chosen because they share distinct characteristics of gestural interactions. These two cases, along with Al-jazzari and Betablocker by Griffiths (2007) and iMac Music by Reus (2012), belong to the category of bottom-up live coding systems.

CodeKlavier CKalcuλator (Table 1) is a performance system that recognises the musical patterns of the pianist-coder. The extracted MIDI patterns correspond to gestural sequences that the pianist performs to write lambda functions, a common construct in functional programming, on-the-fly. The recognition algorithm identifies repetitive melodic patterns and then instructs the lambda functions to perform the corresponding code evaluations. The system affords simple operations, such as addition and multiplication, and anecdotal evidence from the pianist-coder suggests that it is a highly challenging approach to live code, as the mind is split into two tasks simultaneously. The first task corresponds to serial skilled actions resulting in the musical outcome, and the second task is to perform conscious operations to write and modify a running program. Playing the piano indeed involves pre-reflective processes. Then, on the notational level of the generated code, the piano-coder has to allocate cognitive resources that may hinder expressivity during the performance. This approach showcases how musical gestures can be employed to write computer programs. In practice, the pianist plays repetitive melodic patterns and incorporates them into musical improvisation.

When developing the approach to live hardware coding (Diapoulis and Zannos 2012, 2014), the initial motivation was to make live coding somewhat more *humane*. At the time, as a student of material science, I naively imagined material crystal structures that could be
live-coded based on a notation analogous to Miller indices, a notation system used in crystallography. This notation system offers a three-dimensional spatial representation of crystal structures in a binary-like notation. In this manner, elementary building blocks of material structures can be abstracted so that the engineer can imagine larger-scale crystal structures. As a result, what we literally see as a grain of salt can be reduced to a minimal expression, and from the minimal expression, we can deduce some of the material properties such as stiffness and conductance. The stateLogic machine (Table 1) is an experimental interface still in development (Diapoulis et al. 2022) and demonstrates the generation of a formal language from the bottom up. Initially, three input buttons were used, which depended on a clock-based paradigm. In this way, the user could apply stream-based updates within less than 0.5 seconds. These short-duration updates can enable pre-reflective processes. It is possible to automate certain pattern predictions without expending mental effort, such as looping into even and odd numbers. However, the clock-based interaction distorts the perceived immediacy of the interface, especially when the updates are larger than 100 milliseconds (Nash 2012). In principle, Al-jazzari, Betablocker and iMac Music are also clock-based systems. Nevertheless, there are different embodied percepts when the clock rate is fast enough, transforming the interaction into phenomenally continuous percepts.

Technically speaking, these systems may not influence the language development process but they do affect the algorithmic structure of the written programs. We gesturally intervene to modify the algorithm's very workings and there is clearly a minimal distance between the two. For instance, in Al-jazzari and Betablocker, the coder may experiment with multiple instruction sets or update them on-the-fly. Such dynamic updates on the level of an instruction set can account for developing a

[7]By weak temporality I refer to the typing gestures during live coding, which lack any sense of strong temporal couplings.

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

language on-the-fly. This is because all studied systems construct the running program by on-the-fly assemblages of low-level languages. In iMac Music, Reus (2012) is taking a radical stance and rewires the internal hardware components of a personal computer (Table 1). This approach goes one step further and addresses live coding by intervening in the wirings of the hardware. How would live coding look when we can hot-swap modular hardware components? What would previewing algorithms look like? Would the system re-arrange its constituent structure to make a new component? I find a large range of possibilities for human–material interactions (Ishii et al. 2012) in live coding, and it is fascinating that we have not yet scratched the surface of this topic.

5.3. Mixed systems

All studied performances in the mixed category utilise direct manipulation with the mouse to some extent. The most diverse performance is that by The Duchess of Turing, where various systems are used, some of which are non-conventional, and one that can be regarded as a bottom-up system. Specifically, the TOPLAP app (Collins 2015) uses machine listening algorithms to program an instruction set. I present this performance, called Using various, because of the broad variety of systems used. All other performances use a single programming system and direct manipulation. The performance by uiae and redFrik share many similarities, but the difference between textual and visual language is a major difference. Threnoscope by Magnusson presents an important contribution to notation systems while Approximate Programming by Kiefer presents a sophisticated algorithm for on-the-fly adjustments. I selected The Duchess of Turing performance to question whether we can actually form a mental model of the running program using listening and all other systems because of the diverse direct manipulation interactions, ranging from simple parameter adjustments to complicated tree algorithms.

What happens when the coder is using a mixture of different systems? How can the performer form a mental model of the running program(s) by simply listening to the musical outcome? These questions are well illustrated in a live coding performance by The Duchess of Turing (Table 1) for the fifteenth anniversary of TOPLAP. During the performance, the composer-programmer uses a variety of systems, such as PureData (PD), MAX/MSP, CSound and Scheme, and various systems developed by Nick Collins, such as BBCut, Autom8 and TOPLAP app. The live coder switches between systems to generate sound patterns, creating a mashup of live coding systems that resembles DJ-ing or bricolage of live coding systems (McLean and Wiggins 2010a). The software systems are not interconnected; it is rather the musical outcome that connects everything together.

Another interpretation of this performance is that the language design and programming structures no longer matter when rendered to sound, as auditory perception can compensate and perceptually 'bind' everything together. Would that be somehow opposed to the view about the idiomaticity of low-level tools, as presented in McPherson and Tahıroğlu (2020)? Idiomaticity denotes that the design of a tool, regardless of how open-ended it is, carries design constraints that can distinctly shape the musical outcome. This discussion opens questions about agency. The notion of idiomaticity of programming languages looks aligned with a theoretical view of how influential agency progresses from all parties involved, to the spectator of the artwork by creating feedback networks (Dahlstedt 2021). The performance in question appears to suggest that when information is materialised, it loses its affiliation.

One engaging live coding performance by uiae, Using PD (Table 1), demonstrates how gestural interaction with the mouse modifies the system's algorithmic structure. Interestingly, as the different PD objects move around the programming panel, they modify their connectivity, likely from the relative distance between nodes. Fredrik Olofsson (Table 1, superCollider) also demonstrated a similar approach, where direct manipulation using the mouse modified the routing of the audio busses. Another system that uses direct manipulation is the Threnoscope by Thor Magnusson (2014b), which uses a generative system combining graphic scores and text-based programming. Using Threnoscope (Table 1), the coder can continuously control the parameters of the generated drone sounds, using interactive visuals that display a descriptive notation shown as coloured rings.

Chris Kiefer (2015) presented an approach based on a gestural controller called Approximate Programming (Table 1). This is based on genetic algorithms and the exploration of parameter spaces for machine musicianship. Approximate programming is an interactive programming paradigm where the user controls the generation of different unit generators, using a multiparameter gestural controller. The major contribution of Kiefer's work is the dynamic modification of algorithmic tree structures of interconnected unit generators. Such an approach to live coding demonstrates how gestural interactions can be applied to modify the algorithmic structure of a binary tree.

While it can be difficult to distinguish between algorithmic and parameter modifications, I would like to discuss this from a perspective that might clarify how I use the two terms. For a programmer, a variable is a powerful abstraction. In programming

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

environments such as SuperCollider (McCartney 2002), a variable can be a generic box that holds different things. When a system is using variables as numerical parameters, such as a filter's cutoff frequency, then most likely, the generated sound patterns are predictable. If we use a variable to apply simple mappings between a control parameter and a range of numerical values, then we do not exploit the affordances of a variable. Of course, in some cases, simple 1-to-1 mappings are necessary and desirable; for instance, we may want a single knob to control the tempo and nothing more than that. Still, I draw attention to this here to clarify that gestural interactions using the mouse or other input devices may correspond to either simple modifications of a numerical parameter or complicated algorithmic modifications such as structural rearrangements of binary trees (e.g., Kiefer 2015).

This discussion on intervening with gestures on the algorithm itself is not related to *gesture-sound mappings* in the sense of running an algorithm to identify some gestural characteristics and render plausible sounds. I am interested in how we gesturally modify or re-program the algorithms on-the-fly. So, to avoid confusion with gesture-sound mappings, I here focus on *gesture-algorithm mappings*, which share some similarities with dynamic mapping strategies (Dahlstedt 2009; Kiefer 2015).

5.4. Summary of the observations

In this section, I have discussed how meaningful musical gestures can be produced using typing during canonical live coding. Inspired by Baalman's typing automaticity, I question whether gestures can influence the programming language. The answer becomes apparent when I examine the bottom-up systems, where bodily gestures can operate on pre-reflective processes and influence the development of the programming language. An online performance that creates mashups from various programming languages may indicate the importance of sound, not the tool. It is similar to when we have a phase transition in a material (e.g., water crystallises to ice), but the process is not reversible anymore, meaning that when the code is rendered to sound, the process is no longer reversible. Maybe this performance suggests that when the tool is ignorant of its consequences to the environment, it loses its agency.

Several performance systems incorporating direct manipulation have been examined, and I underline the broad spectrum of interactions, from simple manipulations of numbers to sophisticated algorithms operating on binary trees. These systems differ from canonical systems as they offer the user an intuitive manner for interactions. In cases such as Approximate Programming, the live coder can modify the algorithmic structure on-the-fly, using continuous gestural interactions on multiparameter controllers. This also exhibits pre-reflective activations during a performance. On the other hand, such systems may not allow code changes to be monitored. When this happens, we lose the possibility of involuntary imageries due to notational audiation. Thus, there is a fine line between intuitive control and how to monitor informative notations. For instance, in Threnoscope, the live coder can monitor the descriptive notation as interactive visualisations of coloured rings and can continuously control the parametric changes. Thus, if the system's temporal updates on the prescriptive notation are too fast, causing them to become uninformative, we can facilitate the use of descriptive notations instead (Magnusson 2019).

6. CONCLUSIONS

This article has explored whether pre-reflective processes can be activated in live coding. I presented an embodied cognition view to live coding music performance by discussing how notation, while hindering our bodily relationship with the generated sound patterns, also offers some opportunities. A first indication comes from the literature review, where the potential of notational audiation is posited as valuable to live coders due to involuntary musical imagery. Typing on a keyboard is an activity that incorporates serial skilled actions. While this shares similarities with traditional music performance because they are both based on serial skilled actions, typing is performed out of time order, making it difficult to observe entrained processes in canonical live coding. On the other hand, two cases of bottom-up live coding systems (CodeKlavier CKalcuλator and stateLogic machine) exhibit gestural interactions on timeframes that are capable of engaging pre-reflective processes during a performance. Several systems that use direct manipulation are examined, and at least one (Approximate Programming) also exhibits fast processes, although it is questionable whether the user can exploit the prescriptive notation during a performance. Owing to design constraints, using a descriptive notation can be an alternative so that involuntary imageries may be activated due to notational audiation. The study provides a first account of exploring fast processes in live coding by conducting observations from online videos and combining diverse literature, drawing on music psychology and music perception studies. The study is limited to solos, which could potentially constrain the results. But I intentionally limited the study to musician–system interactions and not musician–musician interactions. Thus, some essential parts of the performer's interactivity variations are

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press

identified, such as the temporal unfolding of bodily gestures during a performance and how similar or different this might be from traditional instrumental performance. Future studies could focus on intersubjective experiences in live coding and combine online video observations with interview studies and questionnaires to provide a more holistic view of how practitioners and the community experience fast processes in musical live coding.

This study contributes to the understanding of live coding, focusing on the gestural and psychological aspects of the practice, which have not been extensively studied before. This benefits both the practitioners, to increase understanding of the practice from within the community, and the general electronic music community, which may not be familiar with the particular conditions and practices of live coding. The methodology and theoretical background can be helpful to other under-represented groups practising generative music and autonomous music systems. As such, it may be used as a methodological template in how to conduct observational research from online videos, especially when there is little theoretical background in the field.

Acknowledgements

I thank Palle Dahlstedt for proofreading and supervision. I am also thankful to the anonymous reviewers and guest editors for making this article better.

REFERENCES

American Psychological Association. N.d. Mental Model. *APA Dictionary of Psychology*. https://dictionary.apa.org/mental-model (accessed 23 January 2023).

Baalman, M. 2009. Code LiveCode Live, or Livecode Embodied. [Performance]. *Proceedings of the International Conference on New Interfaces for Musical Expression*, NIME, 329.

Baalman, M. 2015. Embodiment of Code. *Proceedings of the First International Conference on Live Coding*. Leeds: ICLC, 35–40.

Baalman, M. n.d. Code LiveCode Live. https://marijebaalman.eu/projects/code-livecode-live.html (accessed 14 September 2022).

Baily, J. 1999. Ethnomusicological Perspective: Response to Sawyer's 'Improvised Conversations'. *Psychology of Music* **27**(2): 208–11.

Bishop, L., Bailes, F. and Dean, R. T. 2012. Musical Imagery and the Planning of Dynamics and Articulation during Performance. *Music Perception: An Interdisciplinary Journal* **31**(2): 97–117.

Blackwell, A. F. and Collins, N. 2005. The Programming Language as a Musical Instrument. *Proceedings of 17th Psychology of Programming Interest Group*, University of Sussex, 120–30.

Blackwell, A. F., Cocker, E., Cox, G., McLean, A. and Magnusson, T. 2022. *Live Coding: A User's Manual*. Cambridge, MA: MIT Press.

Bovermann, T. and Griffiths, D. 2014. Computation as Material in Live Coding. *Computer Music Journal* **38**(1): 40–53.

Brattico, E., Brattico, P. and Jacobsen, T. 2009. The Origins of the Aesthetic Enjoyment of Music—A Review of the Literature. *Musicae Scientiae* **13**(2_suppl): 15–39.

Brodsky, W., Kessler, Y., Rubinstein, B. S., Ginsborg, J. and Henik, A. 2008. The Mental Representation of Music Notation: Notational Audiation. *Journal of Experimental Psychology: Human Perception and Performance* **34**(2): 427.

Clayton, M., Jakubowski, K., Eerola, T., Keller, P. E., Camurri, A., Volpe, G. and Alborno, P. 2020. Interpersonal Entrainment in Music Performance: Theory, Method, and Model. *Music Perception: An Interdisciplinary Journal* **38**(2): 136–94.

Collins, N. 2015. Live Coding and Machine Listening. *Proceedings of the First International Conference on Live Coding*. Leeds: ICLC, 4–11.

Collins, N. and McLean, A. 2014. Algorave: Live Performance of Algorithmic Electronic Dance Music. *Proceedings of the International Conference on New Interfaces for Musical Expression*. London: NIME: 355–8.

Collins, N., McLean, A., Rohrhuber, J. and Ward, A. 2003. Live Coding in Laptop Performance. *Organised Sound* **8**(3): 321–30.

Dahlstedt, P. 2009. Dynamic Mapping Strategies for Expressive Synthesis Performance and Improvisation. *International Symposium on Computer Music Modeling and Retrieval*. Berlin: Springer, 227–42.

Dahlstedt, P. 2012. Between Material and Ideas: A Process-Based Spatial Model of Artistic Creativity. In J. McCormack and M. d'Inverno (eds.), *Computers and Creativity*. Berlin: Springer, 205–33.

Dahlstedt, P. 2018. Action and Perception: Embodying Algorithms and the Extended Mind. In A. McLean and R. Dean (eds.) *The Oxford Handbook of Algorithmic Music*. Oxford: Oxford University Press, 41–65.

Dahlstedt, P. 2021. Musicking with Algorithms: Thoughts on Artificial Intelligence, Creativity, and Agency. In E. R. Miranda (ed.) *Handbook of Artificial Intelligence for Music*. Cham: Springer, 873–914.

Diapoulis, G. and Dahlstedt, P. 2021a. An Analytical Framework for Musical Live Coding Systems Based on Gestural Interactions in Performance Practices. *Proceedings of the International Conference on Live Coding*. Valdivia, Chile: ICLC.

Diapoulis, G. and Dahlstedt, P. 2021b. The Creative Act of Live Coding Practice in Music Performance. *Proceedings of the 32nd Psychology of Programming Interest Group*, York, UK.

Diapoulis, G. and Zannos, I. 2012. A Minimal Interface for Live Hardware Coding. *Live Interfaces*, ICSRiM, University of Leeds.

Diapoulis, G. and Zannos, I. 2014. Tangibility and low-Level Live Coding. *International Computer Music Conference*. Athens: ICMC&SMC.

Diapoulis, G., Zannos, I., Tatar, K. and Dahlstedt, P. 2022. Bottom-Up Live Coding: Analysis of Continuous Interactions towards Predicting Programming Behaviours. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Auckland: NIME.

Emmerson, S. 2017. *Living Electronic Music*. Abingdon: Routledge.

Godøy, R. I. 2004. Gestural Imagery in the Service of Musical Imagery. *International Gesture Workshop*. Berlin, Heidelberg: Springer.

Godøy, R. I. 2021. Constraint-Based Sound-Motion Objects in Music Performance. *Frontiers in Psychology*, **12**. https://doi.org/10.3389/fpsyg.2021.732729.

Goldman, A. 2019. Live Coding Helps to Distinguish between Embodied and Propositional Improvisation. *Journal of New Music Research* **48**(3): 281–93.

Griffiths, D. 2006. Betablocker. *General Public Licence (GPL)*. www.pawfal.org/flotsam/betablocker/betablocker.scm (accessed 9 May 2023).

Griffiths, D. 2007. Game Pad Live Coding Performance. *Die Welt als virtuelles Environment*. Dresden: TMA Helleraue.

Haga, E. 2008. Correspondences between Music and Body Movement. Doctoral dissertation, University of Oslo.

Haworth, C. 2018. Technology, Creativity, and the Social in Algorithmic Music. In A. McLean and R. Dean (eds.) *The Oxford Handbook of Algorithmic Music*. Oxford: Oxford University Press, 557–81.

Hutchins, C. C. 2015. Live Patch/Live Code. *Proceedings of the First International Conference on Live Coding*. Leeds: ICLC, 147–51.

Ishii, H., Lakatos, D., Bonanni, L. and Labrune, J. B. 2012. Radical Atoms: Beyond Tangible Bits, toward Transformable Materials. *interactions* **19**(1): 38–51.

Jakubowski, K. 2020. Musical Imagery. In A. Abraham (ed.) *The Cambridge Handbook of the Imagination*. Cambridge: Cambridge University Press, 187–206.

Jensenius, A. R. 2022. *Sound Actions: Conceptualizing Musical Instruments*. Cambridge, MA: MIT Press.

Jensenius, A. R., Wanderley, M. M., Godøy, R. I. and Leman, M. 2010. Musical Gestures: Concepts and Methods in Research. In R. I. Godøy and M. Leman (eds.) *Musical Gestures: Sound, Movement, and Meaning*. New York: Routledge, 12–35.

Keller, P. E. 2008. Joint Action in Music Performance. In F. Morganti, A. Carassa and G. Riva (eds.) *Enacting Intersubjectivity: A Cognitive and Social Perspective on the Study of Interactions*. Amsterdam: IOS Press, 205–21.

Keller, P. E. 2012. Mental Imagery in Music Performance: Underlying Mechanisms and Potential Benefits. *Annals of the New York Academy of Sciences* **1252**(1): 206–13.

Keller, P. E. and Appel, M. (2010). Individual Differences, Auditory Imagery, and the Coordination of Body Movements and Sounds in Musical Ensembles. *Music Perception* **28**(1): 27–46.

Kiefer, C. 2015. Approximate Programming: Coding through Gesture and Numerical Processes. *Proceedings of the First International Conference on Live Coding*, ICSRiM, University of Leeds.

Kosslyn, S. M. 1996. *Image and Brain: The Resolution of the Imagery Debate*. Cambridge, MA: MIT Press.

Leman, M. 2007. *Embodied Music Cognition and Mediation Technology*. Cambridge, MA: MIT Press.

Leman, M. 2016. *The Expressive Moment: How Interaction (with Music) Shapes Human Empowerment*. Cambridge, MA: MIT Press.

Leman, M. and Godøy, R. I. 2010. Why Study Musical Gestures?. In R. I. Godøy and M. Leman (eds.) *Musical Gestures: Sound, Movement, and Meaning*. New York: Routledge, 3–11.

Loveday, C., Woy, A. and Conway, M. A. 2020. The Self-Defining Period in Autobiographical Memory: Evidence from a Long-Running Radio Show. *Quarterly Journal of Experimental Psychology* **73**(11): 1969–76.

Magnusson, T. 2011. The ixi lang: A supercollider Parasite for Live Coding. *Proceedings of the International Computer Music Conference*. Huddersfield: ICMC.

Magnusson, T. 2014a. Herding Cats: Observing Live Coding in the Wild. *Computer Music Journal* **38**(1): 8–16.

Magnusson, T. 2014b. Improvising with the Threnoscope: Integrating Code, Hardware, GUI, Network, and Graphic Scores. *Proceedings of the International Conference on New Interfaces for Musical Expression*. London: NIME, 19–22.

Magnusson, T. 2019. *Sonic Writing: Technologies of Material, Symbolic, and Signal Inscriptions*. London: Bloomsbury Academic.

Marklin, R. W. and Simoneau, G. G. 2004. Design Features of Alternative Computer Keyboards: A Review of Experimental Data. *Journal of Orthopaedic & Sports Physical Therapy* **34**(10): 638–49.

McCartney, J. 2002. Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal* **26**(4): 61–8.

McKechnie, L. E. F. 2008. *Unstructured Observation*. In Lisa M. Given (ed.), *The Sage encyclopedia of Qualitative Research Methods*. Thousand Oaks, CA: Sage, 907–8.

McLean, A. 2014. *Stress and Cognitive Load. Collaboration and Learning through Live Coding*. *Report from Dagstuhl Seminar*, **13382**: 145–6.

McLean, A. and Wiggins, G. A. 2010a. Bricolage Programming in the Creative Arts. *Proceedings of the 22nd Psychology of Programming Interest Group*, Madrid.

McLean, A. and Wiggins, G. 2010b. Tidal–Pattern Language for the Live Coding of Music. *Proceedings of the 7th Sound and Music Computing Conference*, SMC, 331–4.

McLean, A. and Wiggins, G. A. 2011. Texture: Visual Notation for Live Coding of Pattern. *Proceedings of* the *2011 International Computer Music Conference*, Huddersfield.

McPherson, M. J. and Limb, C. J. 2019. Improvisation: Experimental Considerations, Results, and Future Directions. In Rentfrow, P. J. and D. J. Levitin (eds.) *Foundations in Music Psychology*. Cambridge, MA: MIT Press.

McPherson, A. and Tahıroğlu, K. 2020. Idiomatic Patterns and Aesthetic Influence in Computer Music Languages. *Organised Sound* **25**(1): 53–63.

Myers, B. 1994. Challenges of HCI Design and Implementation. *Interactions* **1**(1): 73–83.

Nash, C. 2012. Supporting Virtuosity and Flow in Computer Music. Doctoral dissertation, University of Cambridge.
Nilson, C. 2007. Live Coding Practice. *Proceedings of the 7th International Conference on New Interfaces for Musical Expression*. New York: NIME: 112–17.
Nilson, C. 2016. *Collected Rewritings: Live Coding Thoughts, 1968–2015*. Burntwood, UK: Verbose. https://composerprogrammer.com/research/collectedrewritings.pdf
Noriega, F. I. and Veinberg, A. 2019. The Sound of Lambda. *Proceedings of the 7th ACM Sigplan International Workshop on Functional Art, Music, Modeling, and Design*. Berlin: FARM, 56–60.
Palmer, C. 1997. Music Performance. *Annual Review of Psychology* **48**(1): 115–38.
Palmer, C. 2012. 10 Music Performance: Movement and Coordination. In D. Deutsch (ed.) *The Psychology of Music*, 3rd edn. London: Elsevier, 405–22.
Repp, B. H. and Su, Y. H. 2013. Sensorimotor Synchronization: A Review of recent Research (2006–2012). *Psychonomic Bulletin & Review* **20**(3): 403–52.
Reus, J. 2012. iMac Music. https://web.archive.org/web/20161027164504/https://jonathanreus.com/portfolio/imac-music/ (accessed 8 April 2023).
Roberts, C. and Kuchera-Morin, J. 2012. Gibber: Live Coding Audio in the Browser. *Proceedings of the International Computer Music Conference*. Ljubljana: ICMC.
Roberts, C. and Wakefield, G. 2018. Tensions and Techniques in Live Coding Performance. In A. McLean and R. Dean (eds.) *The Oxford Handbook of Algorithmic Music*. Oxford: Oxford University Press, 293–317.
Rohrhuber, J., de Campo, A., Wieser, R., Van Kampen, J. K., Ho, E. and Hölzl, H. 2007. Purloined Letters and Distributed Persons. *Music in the Global Village Conference*, Budapest.
Roos, F. and McLean A. 2023. Strudel: Live Coding Patterns on the Web. *Proceedings of the International Conference on Live Coding*. Utrecht: ICLC.
Salazar, S. and Armitage, J. 2018. Re-engaging the Body and Gesture in Musical Live Coding. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Blacksburg, VA: NIME.
Sayer, T. 2015. Cognition and Improvisation: Some Implications for Live Coding. *Proceedings of the International Conference on Live Coding*. Leeds: ICLC.
Shapiro, L. and S. Spaulding 2021. Embodied Cognition. In E. N. Zalta (ed.) *The Stanford Encyclopedia of Philosophy*, Winter edn. https://plato.stanford.edu/archives/win2021/entries/embodied-cognition/ (accessed 18 January 2023).
Shepard, R. N. and Metzler, J. 1971. Mental Rotation of Three-Dimensional Objects. *Science* **171**(3972): 701–3.
Snee, H. 2013. Making Ethical Decisions in an Online Context: Reflections on Using Blogs to Explore Narratives of Experience. *Methodological Innovations Online* **8**(2): 52–67.
Tanimoto, S. L. 2015. Livesolving: Enabling Collaborative Problem Solvers to Perform at Full Capacity. *Proceedings of the International Conference on Live Coding*. Leeds: ICLC.
Theodosopoulou Bourlogianni, D. 2021. Music and Autobiographical Memory: How an Analysis of Desert Island Discs May Help Conceptualise Personalised Music Interventions for People Living with Dementia. Doctoral dissertation, University of East Anglia.
Varela, F. J., Thompson, E. and Rosch, E. 2017. *The Embodied Mind, Revised Edition: Cognitive Science and Human Experience*. Cambridge, MA: MIT Press.
Zatorre, R. J. and Halpern, A. R. 2005. Mental Concerts: Musical Imagery and Auditory Cortex. *Neuron* **47**(1): 9–12.

VIDEOGRAPHY

McCallum, L. 2011. *Show Us Your Screens. Vimeo*, February 22. https://vimeo.com/20241649 (accessed 14 April 2023).
Villaseñor, H. 2019. Dar forma al espacio: Primer Festival Expresiones Contemporáneas. YouTube, 10 November. https://youtu.be/vARqRuMoPx8 (accessed 14 April 2023).

https://doi.org/10.1017/S1355771823000444 Published online by Cambridge University Press