Disruption and creativity in live coding

Ushini Attanayake*, Ben Swift*, Henry Gardner* and Andrew Sorensen†
*Research School of Computer Science
Australian National University, Canberra, ACT 2600, Australia
Email: {ushini.attanayake,henry.gardner,ben.swift}@anu.edu.au
†MOSO Corporation, Launceston, Tasmania, Australia
Email: andrew@moso.com.au

***Abstract*—Live coding is an artistic performance practice where computer music and graphics are created, displayed and manipulated in front of a live audience together with projections of the (predominantly text based) computer code. Although the pressures of such performance practices are considerable, until now only familiar programming aids (for example syntax highlighting and textual overlays) have been used to assist live coders. In this paper we integrate an intelligent agent into a live coding system and examine how that agent performed in two contrasting modes of operation—as a *recommender*, where on request the the agent would generate (but not execute) domain-appropriate code transformations in the performer's text editor, and as a *disruptor*, where the agent generated and immediately executed changes to the code and the streaming music being performed. A within-subjects study of six live coders suggested that participants of the study preferred the agent's disruptive mode as a mechanism for enhancing the creativity of their live coding performances.**

***Index Terms*—Intelligent agents, Interaction paradigms, Musical instrument digital interfaces, Software support tools**

I. Introduction

Live coding is an exciting and risky form of performative, creative, end-user programming (EUP) [1] where computer music and graphics are created, manipulated and displayed on screens in front of a live audience. In a typical performance, a live coder starts from a blank text editor window and incrementally writes and evaluates code. When the live coder evaluates a block of code, any changes take immediate effect, creating and manipulating the music, visuals or robotic instruments during the performance. Although musicians have long been recognised as potential EUPs [2], [3], the real-time, performative aspect of live coding places demands on the programmer which are unlike other EUP contexts [4], particularly the challenge of creating new and interesting musical ideas in the moment of performance.

Live coders typically use text completion and syntax highlighting facilities of modern text editors to efficiently write clear code, both for the audience and themselves. In addition, domain-specific visual annotations and adornments have been used in some cases, for example overlays which blink in time with the musical output [5]–[9]). This work has shown that these visual adornments can help to better expose the structure and temporal execution of live coding performances.

978-1-7281-6901-9/20/$31.00 ©2020 Crown

There has been a long-standing interest in the development of intelligent agents as collaborators in creative live musical performance [10]–[13]. However, in most cases the musician and the intelligent agent collaborate and interact in the sonic domain, i.e. the musical output of human and agent are mixed together, with both being able to listen and respond to each other. Comparatively few systems explore the possibility of an intelligent agent which interacts with the *instrument* directly, although examples of this exist in the NIME community [14], [15]. The nature of live coding, with the live coder as an EUP building and modifying their code (i.e. their instrument) during a performance, provides an opportunity to explore the possibility of intelligent agents that are able to anticipate and recommend new code blocks to be integrated into live coding performances in real time.

In this paper we report on the development and evaluation of such a system. Using a Markov-chain-based intelligent agent in the context of the Extempore live coding environment [16], with text editor support in the form of a custom Visual Studio Code (VS Code) extension, the agent recommends (and in some settings, executes) code modifications during a live coding performance. We have evaluated the operation of this system in two contrasting live coding settings: one where the agent *recommends* modifications to running code that the performer is able to accept or not and one where the agent *disrupts* a live coding performance by immediately executing its recommendations. Both versions of this tool have been evaluated in a within-subjects study involving six live coders. Somewhat surprisingly, the majority of these live coders reported to have preferred the disruptive agent over the recommending agent in improving their creative ideation.

II. Related Work

The challenge of ideation is not unique to live coding—the balance between structure and surprise is an important factor in musical improvisation more generally [17], [18]. In jazz, for example, Coltrane and Davis deliberately lift themselves out of creative stagnation by introducing 'disruptions and incremental re-orientations' into their performance [19]. In computer music, agents which embody the most invasive interactions come under the class of the accompanying agents or 'colleagues' [10]. Such agents generate music independently of a live musician as a virtual accompanist.

For example, Voyager [11] is a multi-agent system that listens to a musician, learns from them and improvises alongside that musician; in such a system the musician has no direct control in decision-making over the music produced by the agent. Examples where the musician has more control over the sonic environment are Cypher [12] and Band-out-of-box (B.O.B) [13]. Though the agent in B.O.B makes changes directly in the sonic space, the musician and the agent interact on a call-and-response basis, giving the musician more control over the trajectory of the performance since their output does not necessarily overlap with the agent. Cypher not only allows the musician to control the stream of MIDI data fed into its 'listening' component, it also makes the mapping of data between its 'listening' and 'playing' components configurable.

A less invasive form of interaction with computer music instruments is found in Martin et al.'s work where an agent collaborates with a group of musicians by adjusting the interfaces available to them [14]. In this work, the musicians used touch-screen interfaces whose appearance and musical scale could be modified by an external agent when that agent felt it to be appropriate. That paper studied versions of the agent that either mandated or recommended interface changes to the group of collaborating musicians.

As discussed in the introduction, live coding is a high pressure performance art where computer code is written in front of a live audience to generate and modify music, visual media and even robotic instruments on the fly. Once a block of code is evaluated in an editor, it is immediately compiled and scheduled for execution and it is at this stage that the live coders and the audience hear the sonified effects of that code. Unsurprisingly, the high-pressure setting of a live performance has seen live coders commit large sections of their performance to memory and resort to using presets [20], [21]. Some live coders even admit to "rarely having new ideas during a performance" [22]. Well rehearsed, pre-written code may lead to a more polished performance, but is at odds with the improvisational experimentation which the "liveness" of live coding so greatly encourages.

In spite of the active development of code annotations to assist with the comprehension of live coding performances in real time [5]–[9], there does not seem to have been a systematic consideration and evaluation of the use of agents in live coding until the present time. It is standard practice in live coding to write your generative algorithms during the performance and if an agent-based model were to run in the background, it would need to be efficient enough to keep up with the real-time demands of the performance [23]. The complexity of common agent-based models, such as machine learning, may restrict the number of concurrent processes a live coder can run, thus affecting their popularity in the live coding community. Therefore, the trade-off between the sophistication and efficiency of an agent model is particularly important to consider in this context. The use of agents that can predict alternate versions of a program in a live coding system has been termed "tactically predictive liveness" and is the fifth level of liveness identified by Tanimoto [24], [25].

```
;; original code
(:> A 3 0 (play syn1 @1 60 dur)
   '(60 63 65 67))

;; reverse
(:> A 3 0 (play syn1 @1 60 dur)
   (reverse '(60 63 65 67)))

;; invert
(:> A 3 0 (play syn1 @1 60 dur)
   '(60 57 55 53))

;; thin
(:> A 3 0 (play syn1 @1 60 dur)
   '(60 67))

;; repeat
(:> A 3 0 (play syn1 @1 60 dur)
   '(60 63 65 63 65 67))

;; rotate
(:> A 3 0 (play syn1 @1 60 dur)
   (rotate '(60 63 65 67)
   (random '(-3 -1))))
```

Fig. 1. Example of an Extempore language pattern and its transformation under the operations 'reverse', 'inverse', 'thin', 'repeat', 'rotate'. In the "recommendation" version of our agent, the top line of code would be overwritten by the one of the transformed lines return from the agent.

III. Agent Description

The agent developed in this work has been inspired by the incremental re-orientations Davis and Coltrane apply in jazz improvisation. It recognises certain characteristic *patterns* in the Extempore language that are written in terms of lists of midi pitches. The agent simulates a re-orientation by applying transformations (such as "rotation" or "inversion") to the pitch lists that appear to be incremental but distinct from the original code. We illustrate these transformations with respect to Figure 1 where the first line of code sets up a pattern for the midi pitch list `'(60 63 65)` representing the notes C4, D#4 and F4. When executed, the pattern named 'A' plays the list of pitches every '3' beats through the 'syn1' synthesizer at volume '60'; each note having a standard duration of 'dur'. The reverse transformation of this pattern has the pitches played in the reverse sequence and this is achieved with the 'reverse' keyword.

The Extempore pattern language has capabilities for specifying harmony and variation in rhythm by nesting pitches within the list and using underscores for rests—this compact representation allows for quick implementation and modification in a live performance setting.

The agent is represented by a time-homogeneous, first-order Markov Chain model [26]. Each state of the Markov Chain represents a transformation that can be applied to an

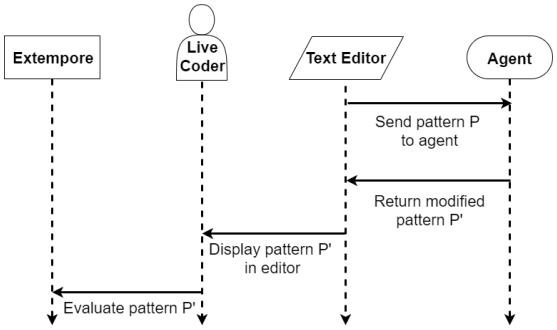

Fig. 2. Recommendation interaction

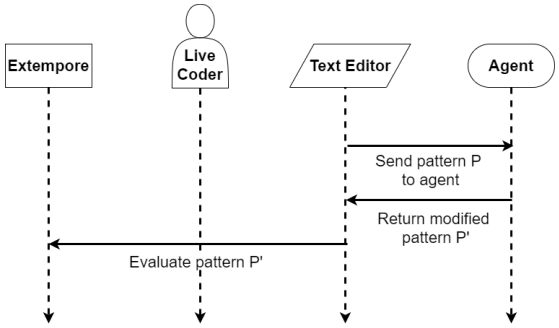

Fig. 3. Disruptive interaction

Extempore pattern pitch list. In addition to 'reverse' (Fig. 1), the agent has four other functions; a) 'Invert', which replaces an increasing interval between adjacent pitches with a decreasing one and vice versa, b) 'Thin', which removes a random number of pitches from the list, c) 'Repeat', repeats a subset of the pitches in the list, d) 'Rotate', moves a random number of pitches from the beginning of the list to the back in a wrap-around fashion. In general, suggestive or 'critiquing' tools are designed based on a set of functions which characterize a specific domain [27]. We limit our agent's capabilities to this specific set of functions; in the domain of music, these functions capture commonly used phrasal transformations [28], making them relevant to a live coder as an EUP.

For a given pattern, the agent stores the previous function it applied and transitions to a new function choice using a probability matrix, the values of which were influenced by music theory and further tuned upon testing within the development team.

A. Recommendation or Disruption

Once a live coder has written a pattern in their editor, they can use hot keys (Ctr+X) to *request* a modification to it from the agent. Upon the coder's request, the VS Code editor extension would momentarily highlight the pattern before sending it to the agent via a TCP connection. In the *recommendation* mode of operation, this triggered the agent to apply a transformation to the pattern's pitch list. The modified pattern was sent back to the editor extension which would overwrite the coder's original pattern. At this stage, the live coder could either choose to a) continue modifying the pattern in the editor, b) evaluate the pattern or c) request another change from the agent. In scenario a) the live coder may choose to add rests to the pitch list the agent generated before using Ctr+Enter to evaluate the pattern. In scenario b), the live coder would use Ctr+Enter to evaluate the agent's pattern without making any changes, while in scenario c), the live coder would use Ctr+X to request another change to that same pattern from the agent. This mode of operation is illustrated in Figure 2 .

In *disruption* mode, once the agent returned the modified pattern, the editor extension directly evaluated and sonified the modified pattern by sending it to the Extempore compiler. In doing this, the live coder could not see the modification the agent made and their original pattern remained unchanged in the text editor. Though the live coder requested a modification and therefore anticipated a change from the agent, the pattern the agent scheduled was unknown to the liver coder. Therefore, the agent was capable of significantly disrupting the trajectory of the performance. The live coder was able to stop the agent's modification from sounding, without stopping any other patterns which were running at the time. Unlike the recommendation mode, the disruption mode did not allow the live coder to modify the pattern generated by the agent, since the code for the pattern did not appear in the editor. This mode of operation is illustrated in Figure 3.

IV. Experiment

We recruited six participants with varying levels of experience as live coders for our experiment. Each participant was asked to perform three improvisational live coding sessions, each session had the agent interacting in one of three degrees of invasion: A "no agent" condition was used as a tutorial and control session and was always undertaken first. Subsequently, the recommendation and disruptive conditions of our agent were experienced in random order for each participant. In each session, participants were told that they could improvise as long as they wanted to up to a cut-off time of 30 minutes (session times ranged from 14 to 30 minutes with six of the 18 sessions lasting for the full 30 minutes). Although there was not a substantial live audience for these sessions, the first author of this paper did their best to appear as an active audience member to put some additional pressure onto the participants. Each session was followed by a short interview and all the changes that the participants made to their code during the sessions was logged. These records and the answers from the interviews were transcribed and a thematic analysis was performed on the interview transcriptions [29]. All participants were experienced musicians. Table I shows their demographics in terms of the nature of their musical

experience and their experience with live coding and the Extempore environment.

ID	Live coding experience	Music experience	First agent
P1	Observed live coding	Computer and Traditional	Recommendation
P2	Experienced with Extempore	Computer and Traditional	Disruptive
P3	No Experience	Traditional	Disruptive
P4	Observed live coding	Traditional	Recommendation
P5	Experienced with live coding	Traditional	Disruptive
P6	No Experience	Traditional	Recommendation

TABLE I
PARTICIPANT DEMOGRAPHICS AND FIRST AGENT EXPERIENCED

The post-session interviews were anchored by a set of questions that directed participants to describe their creative experiences, whether there were any pain points during the sessions and how they perceived their interactions with the agents. Some of these questions were taken from the questionnaire formulated by Martin et al. [14]. The interviews were kept largely free-form in order to allow unanticipated concepts to emerge. When sifting through these interview transcripts, our aim was to look for any signs of variation in idea generation between sessions with the recommendation agent and the disruptive agent. The control session served as a baseline for the intrinsic level of experimentation and ideation that the participants possessed.

V. RESULTS

Variations in "idea generation" observed between the session interviews included changes in the volume of ideas generated and the novelty of the ideas generated (as perceived by the participant). This motivated our selection of codes in the thematic analysis.

A. No Agent

Participants p1, p3, p4 and p6 exhibited a "structured approach" to their improvisation. When asked what the participants did when they felt that the creativity in their music was "plateauing", these four participants indicated they referred to music theory (such as remembering or referring to a chart of midi pitches) to lift them out of such a plateau. These participants were more leisurely in making changes to their code than participants p2 and p5, as demonstrated by relatively long pauses in between evaluating code. In contrast, participants p2 and p5 had a more "experimental" approach to their improvisation, for example p5 stated that their experimentation was aimed at building around a "minor jazz chord progression" by trial and error.

B. Recommendation Agent

The transcripts for the recommendation agent consisted of three major themes. These themes related to (1) the way that participants interacted with the agent, (2) the way the agent interacted with the participants and (3) the way the participants responded to changes made by the agent. We observed that all of the participants "cycled through changes" when interacting with the recommendation agent. This involved requesting changes from the agent repeatedly without evaluating the recommended code. When discussing this, p1, p2, p5 and p6 said that they were interested in seeing what the agent was capable of generating by their repeated requests. Participant p2 felt the changes the agent was making were not *"dramatic"* enough. They clarified that, by this, they wanted the agent to deviate from the existing patterns rather than modify them. By contrast, participants p4, p5 and p6 liked the fact that the agent didn't make any dramatic changes.

Upon observing the participants' response to the changes made by the agent, four participants (p1, p2, p5, p6) recalled specific moments where the change the agent made inspired a new idea and caused them to make subsequent changes to their code. Participant p5 said that they accepted and "leaned into" the agent's recommended changes about 70% of the time. This second theme gave us some evidence that the recommendations introduced by the agent in this mode were capable of stimulating creativity.

The third theme observed during this session was 'disorientation'. Participants found it difficult to navigate the code when the agent replaced their old patterns with its recommended ones. Although the participants were able to revert the changes made by the agent and return to their old pattern, the fact that the agent's modifications occurred in the same line made it particularly hard to monitor changes.

C. Disruptive Agent

It was observed across all participants that *more time elapsed in between requests from the disruptive agent when compared to the recommendation agent*. This could imply that participants took more time to listen to and consider the changes made by the agent because they were presented with the change in their final, sonified state. All participants identified moments where the agent's changes inspired ideas which they later implemented. Even though, p4 said that such moments were rare, all of the other participants felt no anxiety in responding to an agent that could modify the sound in such a disruptive manner. Participants p1, p5 and p6 stated that they took comfort in the fact that they could stop the pattern generated by the agent at any time if they wished.

Four of our participants, (p1, p2, p4, p6) preferred using the disruptive agent and all participants claimed that the disruptive agent triggered a creative response. Participant p6 felt that their preference for the disruptive agent might be due to their ability to better comprehend the changes when hearing them compared with seeing them written in code. Other participants attributed their preference to a greater enjoyment of the music created with the disruptive agent. Participants p3 and p5 preferred the recommendation agent. Both attributed their preference to the ability to be able to see and modify the changes in the code before sonifying that code block.

VI. Conclusion and Future Work

In this study, we observed co-creative interactions between an intelligent agent and six live coders under two modes of operation; recommender and disruptor. Our study provides evidence that a disruptive agent can enhance creativity in live coding performance. Though the disruptive agent gives the live coder less control over the trajectory of the music, all participants in our study felt that their ideas were more creative when interacting with the disruptor agent, while only four participants felt they had creative ideas with the recommender agent. This is a particularly interesting finding given that, excepting the use of a random function, a solo performance rarely involves the live coder relinquishing control to another entity.

We note that the disruptions triggered by our agent were relatively incremental. Future work could test the nature and scale of disruptions that could be tolerated by live coders as well as testing agents against larger cohorts of more experienced live coders and evaluating their use in live performance practice. A larger cohort of participants will also warrant the use of metrics like the Creativity Support Index [30], to strengthen the evaluation of the tool as a computational aid for creative ideation.

Additionally, the effects of hybrid recommender-disruptor interactions on creative ideation could be explored. Two examples are 'in-editor pop-up recommendations', which do not evaluate or replace the original code directly, and systems which replace code in the editor while evaluating it automatically. Both examples could be explored in environments where the musician has complete or no control over triggering the agent's interaction.

Acknowledgments

We would like to thank the participants in our user study, which was conducted under protocol number 2018/497 from the Human Research Ethics Committee at The Australian National University.

References

[1] A. McLean, "Making programming languages to dance to: Live coding with tidal," in *Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design*, 2014, pp. 63–70.

[2] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Computing Surveys*, vol. 43, no. 3, pp. 21:1–21:44, Apr. 2011.

[3] C. Nash, "Manhattan: End-user programming for music," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2014, pp. 221–226.

[4] A. F. Blackwell and S. Aaron, "Craft practices of live coding language design," in *Proc. First International Conference on Live Coding*. Zenodo, 2015.

[5] B. Swift, A. Sorensen, H. Gardner, and J. Hosking, "Visual code annotations for cyberphysical programming," in *2013 1st International Workshop on Live Programming (LIVE)*. IEEE, 2013, pp. 27–30.

[6] C. Roberts, M. Wright, and J. Kuchera-Morin, "Beyond editing: extended interaction with textual code fragments." in *NIME*, 2015, pp. 126–131.

[7] T. Magnusson, "Code scores in live coding practice," in *TENOR 2015: International Conference on Technologies for Music Notation and Representation*, vol. 1, no. 1. Institut de Recherche en Musicologie, 2015, pp. 134–139.

[8] D. Ogborn, E. Tsabary, I. Jarvis, A. Cárdenas, and A. McLean, "Extramuros: making music in a browser-based, language-neutral collaborative live coding environment," in *Proceedings of the First International Conference on Live Coding, University of Leeds, ICSRiM*, 2015, p. 300.

[9] J. Hoffswell, A. Satyanarayan, and J. Heer, "Augmenting code with in situ visualizations to aid program understanding," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[10] T. Lubart, "How can computers be partners in the creative process: classification and commentary on the special issue," *International Journal of Human-Computer Studies*, vol. 63, no. 4-5, pp. 365–369, 2005.

[11] G. E. Lewis, "Too many notes: Computers, complexity and culture in voyager," *Leonardo Music Journal*, pp. 33–39, 2000.

[12] R. Rowe, "Machine listening and composing with cypher," *Computer Music Journal*, vol. 16, no. 1, pp. 43–63, 1992.

[13] B. Thom, "Bob: an interactive improvisational music companion," in *Proceedings of the fourth international conference on Autonomous agents*, 2000, pp. 309–316.

[14] C. Martin, H. Gardner, B. Swift, and M. Martin, "Intelligent agents and networked buttons improve free-improvised ensemble music-making on touch-screens," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 2295–2306. [Online]. Available: https://doi.org/10.1145/2858036.2858269

[15] T. Magnusson, "ixi lang: a supercollider parasite for live coding," in *Proceedings of International Computer Music Conference 2011*. Michigan Publishing, 2011, pp. 503–506.

[16] A. Sorensen and H. Gardner, "Systems level liveness with extempore," in *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2017, pp. 214–228.

[17] D. B. Huron, *Sweet anticipation: Music and the psychology of expectation*. MIT press, 2006.

[18] P. Kivy, *Introduction to a Philosophy of Music*. Clarendon Press, 2002.

[19] F. J. Barrett, "Coda—creativity and improvisation in jazz and organizations: Implications for organizational learning," *Organization science*, vol. 9, no. 5, pp. 605–622, 1998.

[20] M. Cheung, "Reflections on learning live coding as a musician," in *Proceedings of the International Conference on Live Coding (ICLC)*, 2019.

[21] C. Nilson, "Live coding practice," in *Proceedings of the 7th international conference on New interfaces for musical expression*, 2007, pp. 112–117.

[22] A. McLean and G. A. Wiggins, "Live coding towards computational creativity," in *ICCC*, 2010.

[23] A. R. Brown and A. Sorensen, "Interacting with generative music through live coding," *Contemporary Music Review*, vol. 28, no. 1, pp. 17–29, 2009.

[24] 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.

[25] L. Church, E. Söderberg, G. Bracha, and S. Tanimoto, "Liveness becomes entelechy-a scheme for l6," in *The second international conference on live coding*, 2016.

[26] C. Ames, "The markov process as a compositional model: A survey and tutorial," *Leonardo*, pp. 175–187, 1989.

[27] N. Ali, J. Hosking, and J. Grundy, "A taxonomy and mapping of computer-based critiquing tools," *IEEE Transactions on Software Engineering*, vol. 39, no. 11, pp. 1494–1520, 2013.

[28] D. Lewin, *Generalized musical intervals and transformations*. Oxford University Press, USA, 2011.

[29] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[30] E. A. Carroll, C. Latulipe, R. Fung, and M. Terry, "Creativity factor evaluation: towards a standardized survey metric for creativity support," in *Proceedings of the seventh ACM conference on Creativity and cognition*, 2009, pp. 127–136.