LIVE CODING ET/OU MUSIQUE EN « PLAIN-TEXTE »

Raphaël Forment
Université Jean Monnet – ECCLA
raphael.forment@gmail.com

RÉSUMÉ

Le terme de *live-coding* désigne en informatique musicale l'art consistant à programmer à la volée une pièce musicale et/ou visuelle. Cette pratique s'accompagne, le plus souvent, d'une projection pour le public du code manipulé par le musicien. Délimiter les contours de cette pratique constitue un exercice auquel beaucoup se sont livrés par défi ou par jeu. *A minima*, le *live-coding* peut être caractérisé par le fait que les algorithmes au centre de la performance soient écrits et manipulés en temps réel [11], les musiciens relevant le *« défi »* d'entretenir une *« interrelation homme-machine »* le plus souvent improvisée [7]. Le couple éditeur-clavier constitue l'interface de communication première et fondamentale du musicien à son instrument et du musicien à son public. Aujourd'hui en essor, cette pratique peut être perçue comme une expression du désir de retrouver une plus grande proximité et connexion au médium informatique ou comme une volonté d'explorer une approche interactive, computationnelle et algorithmique de la création musicale [6]. La pratique du *live-coding* trouve au travers des *Algoraves* [1], évènements centrés autour de la création algorithmique de musique de danse, sa manifestation la plus visible.

Cet article souhaite étudier la dimension proprement *textuelle* du *live-coding* ainsi que la relation du musicien à son support : le code source ou fichier de texte brut. Nous introduirons un néologisme, celui de musique de ***plain-texte***. Nous jugeons ce terme pertinent pour comprendre la dimension esthétique et politique qui sous-tend cette pratique. Nous essaierons également d'évaluer ce que ce terme peut apporter à une réflexion sur la conception d'outils logiciels dédiés au *live-coding*.

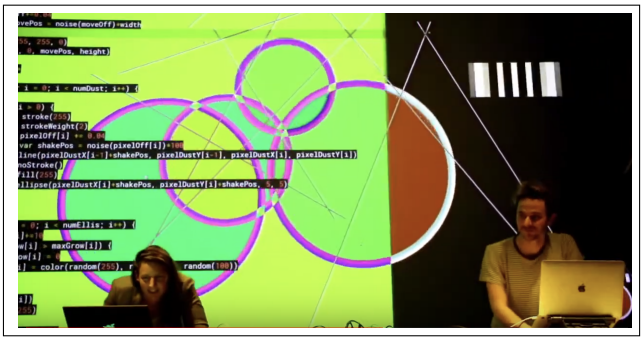

Figure 1. Sabrina Verhage et Jo Kroese (Jobi), La Hague, 22 novembre 2020. Source : **Eulerroom NL_CL #3**.

1. INTRODUCTION

1.1. Présentation

1.1.1. Acteurs de la pratique

La pratique du *live-coding* est aujourd'hui principalement connue au travers des actions de promotion menées par le collectif artistique TOPLAP, pour *Temporary Organisation for the Promotion of Live AudioVisual Programming* [2]. Fondée en 2004 par un groupe d'artistes et chercheurs européens [3], cette organisation a pu servir de catalyseur pour la structuration du renouveau moderne d'une pratique musicale déjà ancienne. L'influence du réseau TOPLAP se lit depuis 2015 au travers de l'organisation de l'*International Conference on Live Coding* (ICLC [4]), principale émanation académique des travaux au sein de la discipline. Elle est également lisible au travers de la popularisation et de la démultiplication, à partir de 2011, des soirées *algoraves*, aujourd'hui organisées dans les principales villes européennes et mondiales.

Les travaux génétiques et historiques récents de Nick Collins [7] et Giovanni Mori [16] ont permis d'établir une double histoire artistique et technique de ce mode de création : oeuvres de la *League of Automatic Music Composers* (1978), devenue *The Hub* à partir de 1983, créations de Pietro Grossi ou Ron Kuivila, outils aujourd'hui délaissés tels que le langage *HMSL* [5]. Ces travaux, parmi d'autres, ont permis de rendre sensibles les liens étroits qui unissent le *live-coding* à des approches et des esthétiques musicales plus anciennes, qu'elles soient algorithmiques, stochastiques, génératives, processuelles et/ou mécanistes. Ces continuités placent la pratique du *live-coding* dans une position de passerelle, la programmation interactive fournissant un outil propice pour l'expérimentation et la formalisation de systèmes compositionnels (en studio, sur scène, comme pratique de recherche/création, etc...).

1 . Mot valise inventé en 2011 par Alex McLean et Nick Collins [8] : *Algorithmic Rave Parties*. Site internet consacré à la recension de ces évènements : `https://algorave.com/`

2 . Site internet et portail du collectif : `https://toplap.org/`

3 . Adrian Ward, Julian Rohrhuber, Fredrik Olofsson, Alex McLean, Dave Griffiths, Nick Collins et Amy Alexander.

4 . Portail de la conférence : `https://iclc.toplap.org/`

5 . *Hierarchical Music Specification Language*, développé au *Mills College* par David Rosenboom, Phil Burk et Larry Polanksy au cours des années 1980.

1.1.2. Nouveaux outils et renouveau

Les nouveaux environnements et langages de CAO ou de synthèse sonore parus au tournant du millénaire, à l'instar de *SuperCollider* (James McCartney, 1996) [13], *ChucK* (Ge Wang, 2003) [30] ou dernièrement *ExTempore* (Andrew Sorensen, 2018) [24], ont chacun placé au centre de leur modèle logiciel la perspective pour l'utilisateur d'interagir en temps réel ou en rétroaction avec le code en cours d'édition : compilation incrémentale, langages dynamiques, capacités d'introspection, *REPLs* [6] . Ce tournant, imputable à des capacités techniques nouvelles et encore inexploitées paraît avoir été également motivé par une série de facteurs externes, plus explicitement esthétiques ou philosophiques.

Perçus comme des outils d'expression directe et spontanée, ces environnements ont rapidement (2004-2018) été exploités comme *instruments* musicaux ou supports pour une nouvelle forme de programmation audio exploratoire, en studio (CAO) aussi bien que sur scène. Le manifeste du collectif TOPLAP, que nous étudierons (*inf.* 2.4.2), constitue une trace de l'enthousiasme alors suscité par cette perspective. Le travail de Julian Rohrhuber sur la plateforme *SuperCollider* fournit l'exemple d'une tendance consistant à penser ces nouvelles plateformes comme des instruments pour la performance. La librairie *JITLib* [21] autorise la redéfinition *à la volée* des graphes audios, le routage dynamique du signal des fonctions audios, l'expérimentation directe sur des algorithmes de synthèse. La librairie de *patterns* et de contrôles offerte par *SuperCollider*, originellement pensée pour la composition algorithmique en temps différé, a ainsi pu être abordée comme un outil dynamique pour l'improvisation, détourné de son mode d'utilisation initial.

De nouveaux *DSLs* à l'intar de *Tidal Cycles* [14], *Sonic Pi* [2] ou *FoxDot* [10], tous trois appuyés sur le moteur *SuperCollider* pour l'exécution sonore, constituent une génération plus récente d'outils explicitement pensés pour l'improvisation et la performance. Ces outils de plus haut-niveau offrent au musicien des abstractions conçues pour lui permettre de se focaliser pleinement sur l'improvisation et le contrôle en automatisant ou simplifiant l'écriture des opérations de bas niveau (routage audio, MIDI et OSC, déclaration des instruments, etc..). Leur syntaxe remarquablement concise permet une expression fluide et facilite la formalisation algorithmique des idées musicales. Apparus au cours de la décennie passée, beaucoup ont été depuis étendus par leurs utilisateurs, portés sur de nouvelles plateformes (*par ex.* le *web audio* [27]) ou personnalisés pour les besoins spécifiques d'une performance ou d'une pratique de composition et de recherche.

1.2. Objectifs

Nous pensons que les musiciens et chercheurs intéressés par la pratique du *live coding* sont unis par le regard singulier qu'ils portent sur le code informatique et sur sa signification humaine, esthétique et symbolique. Ce regard consiste à considérer le code source comme un lieu pour l'expression humaine et comme un espace de dialogue et de communication. Influencés par la *philosophie UNIX* et par l'éthique *hacker* et *maker* [29], les acteurs majeurs de cette discipline et de cet art possédent également une approche particulière de la création et du partage des oeuvres : modularité et partage *open-source* des outils, discussions et ouverture aux contributions d'utilisateurs, etc...

La persistance de la dimension textuelle de cet art, au-delà même de l'évolution et de la mutation des outils (hybridation analogique-digital, instruments embarqués), nous paraît constituer un marqueur de la singularité de cette pratique artistique. La variété stylistique des productions *live codées*, par ailleurs, pourrait suffire à convaincre que le terme de *live coding* ne peut servir à designer une école particulière de composition ou un marqueur stylistique pertinent.

Il s'agit donc d'un rapport singulier au code qui motive les musiciens à adopter cette pratique, à perçevoir dans le *live coding* un rapport nouveau aux outils de création musicale informatique traditionnels. Nous pensons, en conséquent, qu'il est utile de fonder le néologisme de musique de ***plain-texte*** pour accompagner cette observation. Ce néologisme, au contraire du terme générique de *live coding*, insiste pleinement sur la dimension textuelle de la pratique et non sur sa seule spécificité technique. La définition de ce terme nous permettra de préciser les enjeux et les contraintes exercées par le matériau textuel qui supporte l'action musicienne.

2. LE PLAIN-TEXTE

2.1. Précisions étymologiques

```
32 defaultCuts
33 atmos >> stableScalableDensity 0.15 >> defaultCuts >> newKitPattern "kitVarPerc" 6 33  >>
   newEnsPattern "ensAtmoSFA01" 16 37  >> setPs2 "LA16142" "LA16143" >> solo "atmo stab2"
34 playConductors8
35 floodScreen 600 "Let me tell you about what you are looking at. "
36 floodScreen 600 "On the left side of the screen is the interpreter. "
37 floodScreen 600 "That's where the code that I write is executed. "
38 floodScreen 600 "It's the home, so to speak, of the programming language. "
39 floodScreen 600 "Here on the right side of the screen is the text editor. "
40 floodScreen 600 "It's like my piano, my guitar, my body even, inside the computer. "
41
42 playLowend
```

Figure 2. Performance du *live-coder* Renick Bell avec la librairie *Conductive*. Préparation de messages à l'intention du public. Moscou, 15 mai 2020. Source : **Renick Bell, live coding + text**.

Le néologisme « *plain-texte* » est un emprunt direct à l'anglais *plain text*, généralement traduit en français sous le terme de « texte brut » ou de texte sans formattage. Pourquoi, dès lors, ne pas préférer ce terme ? La traduction, en insistant sur l'aspect « brut », perd plusieurs des connotations qui affleurent dans le terme anglais. Celui-ci désigne un matériau plat, sans décoration, uniforme, entièrement visible et compréhensible. [26] *"Plain"* se rat-

6 . *Read, Eval, Print, Loop* : environnement de programmation interactif en lignes de commande. Chaque commande de l'utilisateur est interprétée, et son résultat immédiatement affiché à l'écran.

tache, étymologiquement, à l'idée de platitude, d'uniformité et de continuité. *"Plain text"* désigne une forme basique du texte, un alignement continu et uniforme de caractères plutôt qu'un matériau non travaillé comme pourrait le suggérer l'adjectif « brut ». Une musique en *plain-texte* est par conséquent une musique dont le travail de définition, de composition et d'exécution s'offre au regard, à condition de pouvoir interpréter le symbolisme ou le langage utilisés [7].

Nous faisons le choix de parler de musique de *plain-texte* pour désigner les pratiques musicales qui font de la manipulation du texte une composante importante de la démarche compositionnelle ou du jeu musical. Ce texte peut être utilisé alternativement comme support de stockage, comme support fixe de la partition, comme outil pour la performance, tout en conservant son rôle de témoin et d'objet communiquant par sa projection au cours de la performance.

2.2. Plain texte ou programmation à la volée

Le terme de *live coding* désigne d'ordinaire, hors du domaine musical, une technique de développement logiciel. *Programmer à la volée* consiste à produire par petits ajouts successifs un service ou un logiciel qui sera considéré, *in fine*, comme objet final. *A contrario*, le *live coding* ou musique en ***plain-texte***, par sa nature même, n'est pas une technique orientée vers la production d'un artefact ou d'une oeuvre sur support figé. Le langage, outil du *live coder*, est manipulé et utilisé dans un but exploratoire. L'improvisation consiste, dans ce cadre, à explorer les potentialités ouvertes par la manipulation *dans le temps* d'un matériau musical ou d'un algorithme. En mêlant programmation et exécution simultanée d'un code toujours mouvant, la musique de *plain-texte* invite à se questionner sur la pertinence ou la validité – dans ce cadre – de la notion de programme, d'objet logiciel destiné à une exécution ultérieure [22].

Le geste du *live coder* pourrait se résumer, en essence, à une pratique de *hot-swapping* [8], de composition de fonction ou d'altération progressive d'un texte détaillant les matériaux de la performance et leur transformation. Le terme de musique de *plain-texte* nous semble lever l'ambiguité et la confusion entre une pratique informatique répandue et non musicale et une forme plus spécifiquement artistique de son utilisation. En insistant sur la dimension textuelle, nous distinguons également cette forme de musique électronique improvisée d'autres formes apparentées, telles que le *live patching* ou les *live electronics*, pouvant comporter une part de *hardware hacking*. [9]

2.3. Aspect technique et technologique

Le premier aspect qui nous motive à fonder ce néologisme est à chercher dans la spécificité technique du support de travail du *live coder* : le texte brut. Le code informatique est d'ordinaire stocké et transmis à un compilateur ou un interpréteur sous la forme de texte brut. Ce format, pensé pour sa robustesse et son économie, se révèle particulièrement adapté pour la communication de l'information en réseau (léger, uniforme, strictement défini) ou pour la manipulation directe des sources. La seule condition technique à son déchiffrement est une connaissance préalable de son encodage, lui-même défini par des normes strictes : ASCII, UTF-8, etc... Cette forme de texte constitue historiquement l'un des fondamentaux du stockage et du traitement informatique de l'information.

Le format texte se voit d'ordinaire préféré pour toute tâche requiérant une intervention humaine, un facteur humain dans la démarche de computation. L'absence de fichiers binaires ou de formats non-lisibles permet au musicien d'intervenir à chaque étape de la computation, de la synthèse au contrôle. Pour certains auteurs, à l'instar de Palle Dahlstedt, c'est cette spécificité qui permet de distinguer le *live-coding* comme une pratique dans laquelle le musicien n'improvise pas *« sur »* ou *« avec »* l'algorithme, mais *« dans »* l'algorithme : « *Dans un contexte plus restreint, le musicien est partie prenante d'un système algorithmique temps-réel [...] quelque chose qui est perçu comme quelque chose dans laquelle et de laquelle on devient une partie »* [10] [9]. Parfaitement lisible et uniforme, le format texte permet une forme de transparence et de dialogue entre la source et son utilisateur, souvent absente ou dissimulée par d'autres outils ou instruments augmentés, algorithmiques ou digitaux.

Le texte brut dans la pratique du *live coding* constitue également un support universel. Même un exolangage comme *ORCA* [11], qui représente le code et la computation sous la forme d'une grille bidimensionnelle d'opérateurs et de données, fait appel à ce format tant pour le partage des *patchs* (au format *.txt*) que pour servir d'interface musicale centrale. Ce minimalisme du support possède de nombreux avantages. Son économie et son ouverture permet à la musique en *plain-texte* d'être pratiquée à partir d'un simple éditeur de texte couplé à un interpréteur (*Vim*, *Emacs*, *VSCode*, *Atom*, etc..). Cette même caractéristique permet à un outil de *live coding* de s'intégrer aisément comme séquenceur ou extension d'un instrument hybride (communication *SSH*, messages *OSC*). *ORCA*, originellement programmé en JavaScript à l'aide du *framework* Electron, possède aujourd'hui plusieurs portages en C89, très économes en ressources. Ces derniers permettent à ce langage d'être utilisé sur un micro-ordinateur de type *Raspberry Pi* ou *Monome Norns*.

7 . Le terme « plain » possède déjà une utilisation dans le domaine musicologique, visible au travers du terme de « plain-chant ». Nous ne souhaitons pas y faire référence.

8 . Littéralement, « remplacement à chaud » d'un élément du code source au cours de l'exécution du programme.

9 . Le lien entre *live-coding* et *live-patching* peut se révéler particulièrement difficile à définir. *Scheme For Max* (Iain Duncan, 2020) ou *Gibberwocky* (Charlie Roberts, 2016-2017) en témoignent.

10 . *« In a more tight-knit situation, the musician is part of a real-time algorithmic system [...] that it is perceived as something you step into and become a part of, just as you do in a free improvisation setting ».*

11 . Orca est défini par son concepteur, Devine Lu Linvega, comme « un langage de programmation ésotérique conçu pour créer rapidement des séquenceurs procéduraux (traduction personnelle) » : `https:`

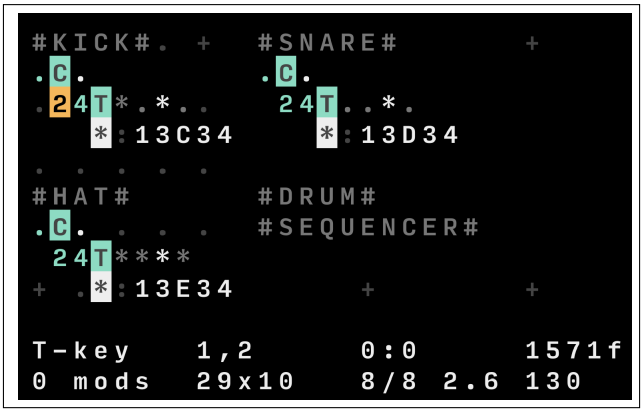

Figure 3. Patch *ORCA* : trois séquenceurs MIDI linéaires.

L'utilisation de texte brut peut paraître au premier abord inadéquate pour saisir, au fil de la performance, l'état du système ou les résultats d'un algorithme en cours d'exécution. Les travaux de Charlie Roberts sur l'environnement et *DSL* JavaScript *Gibber* [19] témoignent pourtant du fait qu'une coloration syntaxique adaptée couplée à une augmentation du texte par l'éditeur offre un résultat proche de celui que pourrait fournir une interface graphique dédiée.

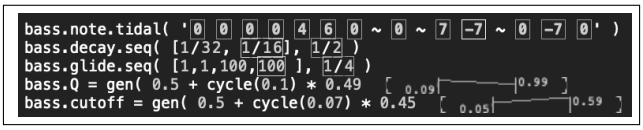

Figure 4. Animation syntaxique dynamique au sein de l'éditeur du langage *Gibber* : `https://gibber.cc/alpha/playground/`

2.4. Une philosophie informatique du texte brut

2.4.1. Philosophie Unix : bonnes pratiques et axiologie

Un second aspect nous paraît justifier la création de ce néologisme : l'existence d'une culture informatique spécifique privilégiant le texte brut comme support de représentation, d'échange et de communication. Les principes de la *philosophie Unix* sont aujourd'hui bien connus [18] et exercent une influence sensible dans le domaine du développement logiciel ou de la création d'interfaces. Cette même influence se révèle tout aussi sensible sur les acteurs majeurs à l'origine du développement moderne du *live coding*. À ce titre, cet art constitue selon nous l'une des passerelles les plus explicites du lien existant entre le domaine de l'informatique musicale et certains principes philosophiques tirés du développement UNIX. Non seulement le langage de programmation constitue l'interface première, mais la relation homme-machine elle-même s'établit autour d'un dialogue ou d'une boucle de rétroaction typique de la ligne de commande et des environnements interactifs UNIX. Ge Wang définit ainsi le langage *ChucK*, par le biais d'un jeu de mot, comme *« a strongly-timed and on-the-fly environ/mentality »* [30]. Alex McLean, créateur de *Tidal* et *live coder* influent, déclare quant à lui en 2007 [1] :

J'utilise le shell UNIX parce que je pense en texte. C'est rapide, et il y a une correspondance magnifique entre le code et les données. C'est facile de rappeller et de mémoriser les actions passées ; vous n'avez pas à vous répéter tout le temps comme dans une fenêtre graphique. Lorsque les environnements en ligne de commande furent développés, ils étaient nommés « langages conversationnels », partie prenante d'un champ de recherche nommé « informatique conversationnelle ». C'est une honte que cette terminologie soit tombée hors d'usage [12] .

Ces propos sont à mettre en lien avec les déclarations aujourd'hui proverbiales concernant la nature ou la philosophie des systèmes UNIX [18] :

C'est la philosophie Unix : écrivez des programmes pour faire une chose et la faire bien. Écrivez des programmes qui fonctionnent entre eux. Écrivez des programmes qui gèrent des flux textuels, car le texte est une interface universelle [13] .

L'avantage du néologisme de *plain-texte* est de permettre de clarifier cette filiation et cet héritage conceptuel. Le texte brut doit être perçu comme un environnement cognitif et mental particulier, comme un mode de représentation qui par son abstraction permet au musicien de se concentrer sur l'expression et la formalisation de sa pensée tout en constituant un support universel de communication. En montrant les sources au cours de la performance, les musiciens de *live coding* font de la performance elle-même un objet et un artefact *open-source*. Le geste, tout autant que l'interaction homme-machine, sont offerts au regard du public et librement reproductibles.

Produire de la musique à l'aide des mêmes outils ouverts et modulaires que ceux utilisés pour la programmation *open-source*, pour le *hacking* (au sens originel) relève également d'une posture axiologique particulière que les travaux de Nicolas Donin et de Baptiste Bacot [3] ont déjà entrepris d'analyser, relevant l'existence de *substrats communs* entre culture du *hacking* et scènes musicales électroniques. La pratique du *live coding*, par son recours au texte, encourage le musicien à percevoir *« l'ordinateur comme [un] objet technique à la fois ouvert et indéterminé ; le code comme [un] support de l'information ; le réseau comme [une] structure de communication »*.

2.4.2. Un cas particulier : le manifeste TOPLAP

Lors de sa création en 2004, le collectif TOPLAP s'est doté d'un manifeste définissant les actions et les pratiques

`//github.com/hundredrabbits/Orca`

12 . *« I use the UNIX shell because I think in text. It's fast, there's a beautiful relationship between data and code, and it's easy to recall and modify past actions – you don't have to repeat yourself all the time like with GUIs. When interactive commandline shells were first developed they were called "conversational languages," part of a field of research called "conversational computing." It's a shame that this terminology fell out of use »*.

13 . *« This is the Unix philosophy : Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface »*.

promues et encouragées par ses membres [31]. Celui-ci semble conçu sur le modèle des grands manifestes artistiques qui ont émaillé l'histoire politique et artistique du XXème siècle. Programmatique et emphatique (peut être à dessein), ce manifeste demeure aujourd'hui une référence lointaine, définissant l'esprit plutôt que la lettre de ce que peut ou doit être une performance reconnue par le collectif. Sa lecture au regard des principes de la philosophie UNIX apporte des éléments de réflexion intéressants pour la définition du néologisme de musique de *plain-texte* : *« Give us access to the performer's mind, to the whole human instrument [...] Obscurantism is dangerous. Show us your screen. [...] The program is to be transcended – Language is the way [...] Algorithms are thoughts [...]* [14] *»*. La déclaration contient également de nombreuses allusions au *plain-texte*, le code de la performance : *« Code should be seen as well as heard »* de même qu'un jeu de mot révélateur : *« skillful extemporisation of algorithm as an impressive display of mental dexterity [...] Live coding may be accompanied by an impressive display of manual dexterity and the glorification of the typing interface »*. En jouant sur la confusion entre dextérité *mentale* et dextérité *manuelle*, les auteurs du manifeste insistent sur le fait que la manipulation du texte est centrale et constitutive de cet art.

Ce manifeste fait également écho à un autre document plus ancien, intitulé *The Generative Manifesto* [15], rédigé par Adrian Ward et Alex McLean (formant le duo *Slub*) à l'occasion d'une présentation à l'*Institute for Contemporary Arts* à Londres en 2000. Si la question du rapport au texte n'est mentionnée que de manière superficielle, ce document constitue certainement une source pour le futur manifeste TOPLAP : *« code reflects human activity [...] Open process, open minds - we have nothing to hide (code is unambiguous, it never hide behind obscurity) [...] Only use software applications written by ourselves »*. Les mêmes thématiques, nimbées de connotations éthiques et politiques, se retrouvent dans ces deux textes fondateurs d'une éthique et d'une esthétique de la pratique du *live coding*.

Ces deux documents sont des témoignages parlants de l'état d'esprit partagé par les acteurs majeurs du domaine. Ge Wang signale dès l'incipit de son travail de thèse que *« les langages de programmation ont peut-être servi comme l'interface la plus intime, la plus précise et la plus générale entre l'homme et l'ordinateur »* [30] et que ces mêmes langages *« peuvent apporter à leurs utilisateurs plus d'expressivité, une plus grande concision et peut-être même une nouvelle manière de penser* [16] *»* (p. 3). Andrew Sorensen, créateur du langage *Ex Tempore* considère quant à lui que *« [la proximité avec le langage] a réintroduit de l'intimité dans le processus et une connexion avec le médium* [17] *»* [5], et que le *live coding* a pour effet de forcer *« une relation plus fondamentale et plus proche à la structure »*. Pour Sam Aaron, créateur de la plateforme de *live coding Sonic Pi*, le *live coding* peut jouer un rôle pour l'apprentissage simultané de l'algorithmique et de la composition musicale à un jeune public : *« nous suggérons que la création de patterns à l'aide d'un langage fonctionnel est un moyen naturel d'enseigner la pensée computationnelle* [18] *»* (p. 35) [2].

Tout ces chercheurs et musiciens *live coders* perçoivent une continuité entre langages de programmation et langages symboliques ou naturels, entre computation et état mental ou expression du flux continu de la pensée musicienne. La question de l'*obscurantisme* est ici étroitement liée à la question des interfaces. La musique de *plain-texte* emploie le code source non par refus des interfaces mais comme substitut au modèle des interfaces graphiques, accusées de voiler ou de dissimuler le propre de l'action et de la pensée musicienne, de ralentir l'expression musicale. Pour ces raisons, le *live coding* en tant que champ académique est parfois perçu à juste titre comme un terrain d'expérimentation propice pour la création de nouvelles interfaces homme-machine dégagées de la métaphore du bureau ou de la fenêtre [4], héritée des travaux d'Alan Kay et du *Xerox Park* au cours des années 1980. Comme en témoigne l'exemple fourni par Gibber (cf. figure 4), le code se doit de fournir une interface suffisante, transparente et lisible.

2.4.3. Le code comme interface expressive

```
d1
  $ rarely (superimpose (jux rev))
  $ every 8 (hurry 2)
  $ stack [s "bd([4|2|6],8)"
          ,s "hh([6|8],8)"
          ,s "sn([1|2],8)"]
```

Figure 5. Rythmes complexes avec *Tidal Cycles* : transformations algorithmiques d'un pattern rythmique (échantillons sonores).

Cette dernière idée transparaît dans la syntaxe offerte par la librairie Haskell *Tidal Cycles*. Profitant de la concision de ce langage fonctionnel, *Tidal* propose un nombre important de fonctions au nom évocateur et rapidement déchiffrables / composables par le musicien : *rarely* pour une application de fonction probabiliste (25% de chance), *stack* pour un empilement d'évènements, *hurry* pour une accélération du pattern et de la vitesse de lecture des échantillons sonores, etc... La boîte à rythme euclidienne de

14 . Nous avons fait le choix de conserver ici le texte anglais : le choix précis des termes et la concision de l'expression nous semblent essentiels pour saisir l'*esprit* de ce manifeste.

15 . *The Generative Manifesto* : `https://slab.org/the-generative-manifesto-august-2000/`

16 . *« To this end, the programming language has perhaps served as the most general, and yet most precise and intimate interface between humans and computers. [...] additional expressiveness, concsieness, and perhaps even different ways of thinking to their users »*.

17 . *[...] « have reintroduced an intimacy of process and a connection with medium [...] or possibly due to our perception of a closer fundamental relationship to structure »*

18 . *« we suggest that making patterns with functional language is a natural way to teach computational thinking »*

la figure 5 est non seulement définie en moins de trois lignes, mais comporte déjà un nombre conséquent de modulations temporelles complexes (accélérations, superpositions, inversions) sur la lecture des échantillons sonores *bd*, *sn* et *hh*. Un public investi se trouve toujours en mesure de déchiffrer, au moins par intuition, le code source offert à son regard.

Par comparaison, la syntaxe de *SuperCollider*, proche de celle du *SmallTalk* ou du *C++*, se révèle beaucoup plus lourde et peu accessible, car de plus bas niveau. Cette dernière se révèle pourtant tout aussi flexible et offre une grande profondeur de contrôle : définition de graphes audios, de synthétiseurs réutilisables, extensibilité du langage et des *UGens*. La librairie *JITLib* (évoquée en *1.1.2*) permet toutefois d'individualiser chaque élément ou module d'une chaîne audio (ou chaîne de contrôle) en les stockant dans une variable de l'espace global. Ce mode de représentation, favorisé par les *live coders*, fait du langage la métaphore d'un instrument modulaire, mode d'intéraction souvent familier pour un musicien électronique.

```
(
// initialisation du serveur audio
p = ProxySpace.push(s.boot);
p.fadeTime = 0.5;
)

(
// percussion modulaire
~modulator = {SinOsc.ar(50).exprange(100,2000)};
~osc       = {SinOsc.ar(~modulator)};
~filter    = {RLPF.ar(
                in:   ~osc,
                freq: LFNoise0.kr(4).range(100,800),
                rq:   SinOsc.ar(1).range(0.2, 0.4))};
~env       = {EnvGen.ar(Env.perc(0.01,0.5),Impulse.kr(2))};
~synth     = {~filter.ar * ~env.ar};
~reverb    = {GVerb.ar(
                in: ~synth, roomsize: 15,
                damping: 0.8) * 0.15};
~reverb.play;
)
```

Figure 6. Utilisation de *ProxySpace* (*JITLib*) pour la définition d'une percussion modulaire modifiable en temps réel. Chaque fonction de la chaîne est individualisée et substituable.

3. LES PARADOXES DE LA TRANSPARENCE

Nous avons essayé, au cours de la section précédente, de définir brièvement le néologisme de musique de *plain-texte*. Ce faisant, nous avons essentiellement discuté de la manière dont est perçu le support textuel par les acteurs de la pratique du *live coding* : ouvert, transparent, communicatif, lisible. Nous avons également accordé de l'attention à la spécificité technique du support, ce qui nous a permis de distinguer le *live coding* d'un certain nombre de pratiques musicales algorithmiques apparentées. L'indétermination de la page blanche et les possibilités expressives ouvertes par la manipulation en temps réel d'un langage de programmation distinguent clairement le *live coding* d'autres pratiques utilisant l'algorithme comme outil et non comme support ou matériau.

Le néologisme de musique de *plain-texte* va nous permettre, à présent, de mettre en valeur certains aspects paradoxaux de l'utilisation du support texte comme interface transparente et intuitive pour l'expression musicale. Préférer ce support pour ses qualités intrinsèques entraîne l'apparition d'un certain nombre de contraintes et de compromis concernant (1) la lisibilité du texte de la performance pour le public et le musicien, (2) l'accessibilité de la pratique et (3) la capacité à noter exhaustivement une performance en *plain-texte*.

L'idéal de transparence et d'ouverture qui constitue l'un des soubassements théorique et esthétique de la discipline (*cf.* 2.4.1-2.4.2) se heurte en pratique à une série de paradoxes. Le refus des interfaces graphiques, le soucis de présenter au public un code source intelligible de la performance et l'expressivité d'un langage de programmation n'entraîne pas mécaniquement un gain de lisibilité ou de clarté pour le public comme pour le musicien. La liberté offerte par le recours à un langage de programmation et la modularité des outils n'offre pas non plus nécessairement plus de liberté pour exprimer et implémenter en temps réel certaines idées musicales.

Ces points constituent aujourd'hui des zones de tension sensibles de la discipline, principalement liées au support du *plain-texte* et aux questionnements qu'il fait naître tant dans la perspective de la pratique musicale que dans celle de l'amélioration des outils : comment rendre plus accessible la pratique du *live-coding* ? comment partager au mieux l'action du musicien et le code source de la performance ? comment le conserver et le transmettre ?

3.1. La manipulation mécanique du texte

Le texte de la performance peut être perçu comme un support mais aussi comme un obstacle pour le musicien. La relation entretenue avec ce dernier se déploie de manière analogue à celle qui unit le musicien à son instrument. En cela, la musique en *plain-texte* rend saillantes un certain nombre de problématiques classiques liées à la conception d'insruments, d'interfaces ou à leur analyse. L'influence de la pratique instrumentale sur la psychologie et les réflexes de l'exécutant est un phénomène déjà largement connu et documenté [25], et déjà étudié pour le cas précis du *live-coding* [23]. De manière similaire, l'influence spécifique d'un langage de programmation musical sur les résultats artistiques de la pratique est aujourd'hui reconnue. Andrew McPherson et Koray Tahiroğlu, suggèrent à cet égard que « *l'idiomaticité est un prisme utile pour considérer aussi bien l'influence des langages que des instruments* [19] » [15]. Chaque instrument ou interface prescrit (volontairement ou non), des gestes et des pratiques d'utilisation privilégiées que le musicien acquiert par le travail ou par l'habitude, et qui viennent influencer son geste et son action.

L'influence de la manipulation en temps réel d'un langage de programmation à des fins performatives reste pourtant encore peu étudiée. Au-delà de l'apprentissage du lan-

19 . *« We suggest that idiomaticity is a useful lens through which to view the influence of both instruments and languages ».*

gage, le *live-coding* exige du musicien une habileté mécanique d'ordinaire absente d'autres pratiques musicales informatiques. La vitesse de frappe et l'art de manipuler un code source non-linéaire (interprété bloc par bloc) constituent des compétences essentielles qu'il s'agit de maîtriser. Ce point a pu encourager certains musiciens, à l'instar de Click Nilson (alter-ego sardonique de Nick Collins), à proposer de purs exercices d'exécution manuelle pour *live coders* : « *À l'imitation de Czerny, de Hanon et de Kreutzer, des exercices de pratique sont fournis ci-dessous pour différentes formes de* live-coding » [20] [17].

La contrainte d'exécution peut influencer le déroulement ou la forme choisie pour une performance, et donc, les résultats artistiques de la pratique. Une vitesse de frappe peu élevée ou imprécise conduit mécaniquement l'improvisateur à se montrer plus prévoyant : préparation d'extraits pré-codés, édition limitée du code ou absence de prise de risque. Seule la pratique permet d'acquérir une mémoire mécanique et d'internaliser certains gestes techniques ou répétitifs. Au-delà de la seule dimension typographique, *live coder* revient également à mémoriser certains *patterns* ou certaines constructions syntaxiques utiles et à pouvoir les convoquer sans erreur et en vitesse. Toute imprécision ou toute mauvaise construction syntaxique résulte, invariablement, en un échec de l'interpréteur et à une nécessaire correction. Ces erreurs lors de la pratique se traduisent parfois en une paralysie ou une stagnation de la performance improvisée. La gestion du risque, déjà largement analysée par les acteurs de la discipline [20], déborde aujourd'hui la seule question de l'habileté mécanique. L'injonction à la vitesse et à la précision incite à la création de langages plus succincts et plus expressifs, produisant un texte plus manipulable mais également plus abstrait (*cf.* figure 5, figure 3).

L'utilisation d'un éditeur de texte modal (*Vim*, *Emacs* (+ *Evil mode*), etc..) peut contribuer à accroître la vitesse et la précision du musicien. Des compétences en programmation et une habitude du développement constituent donc des avantages encore peu commentés et documentés : habitude de naviguer dans les sources, travail sur du code complexe, utilisation de *multiplexers* et outils de complétion syntaxique, etc... La configuration d'un éditeur de texte généraliste pour la pratique musicale s'apparente dès lors à l'entretien d'un instrument. La personnalisation de ce dernier se trouve parfois étroitement corrélée aux capacités expressives du musicien [21]. La transparence de la musique en *plain-texte* possède donc un prix : celui de l'accessibilité et de l'acquisition de compétences techniques et mécaniques. La notion de *plain-texte* invite à considérer la dimension mécanique de la manipulation du texte, qui constitue parfois, selon nos observations, un frein à l'adoption et à la popularisation de la pratique [22].

20 . *« Following Czerny, Hanon and Kreutzer, practice exercises are provided below for live coding of various kinds [...] »*

21 . Des exemples poussés de configuration d'un éditeur de texte existent, à l'instar de la configuration *Emacs* de Sam Aaron (*Sonic Pi*), largement partagée sur la plateforme *GitHub* : `https://github.com/overtone/emacs-live`

22 . Certains langages tels que *FoxDot*, *Sonic Pi* ou *Orca* possèdent leur propre éditeur spécialisé, plus ou moins extensibles, personnalisables et adaptés au langage utilisé.

3.2. Le partage du texte

3.2.1. Projection et lisibilité

La projection du code sur un écran peut être perçue par le public ou le musicien comme une déclaration d'intention, comme un geste esthétique ou comme un gage d'honnêteté. Par son abstraction et l'absence de métaphores graphiques, le code manipulé par le *live-coder* demeure particulièrement hermétique et contre-intuitif pour un public non-initié, à rebours des idéaux du manifeste TOPLAP. Cette paradoxale incommunicabilité du texte algorithmique de la performance fait l'objet de détournements, de pied-de-nez et de ré-appropriations esthétiques. La lisibilité du texte se trouve régulièrement délaissée au profit de la création d'une performance multimédia plus riche dans laquelle le texte s'intègre et trouve un sens.

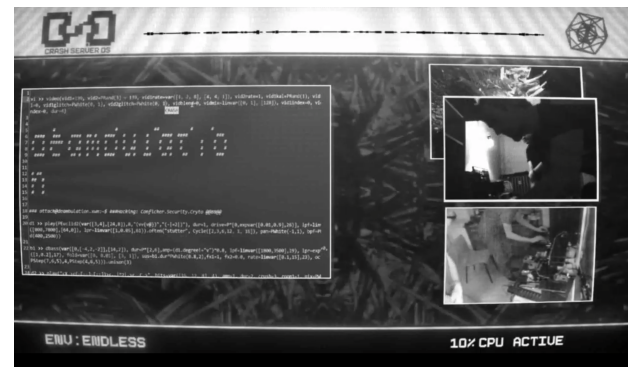

Figure 7. Le duo de *live coding* strasbourgeois *Crash Server* se sert de la projection du code comme élément au sein d'une performance musicale et visuelle rétro-futuriste esthétisée (*ASCII art*, *shaders* génératifs). Source : `https://www.youtube.com/watch?v=jFNlks4ml4Y`

Live-coding visuel génératif et *live coding* musical se retrouvent aujourd'hui étroitement associés. Ces deux tâches sont parfois remplies par le même musicien, qui constitue, ce faisant, une performance multimédia complète. Cette dimension visuelle fait de cette scène un lieu de rencontre pour la promotion des arts algorithmiques de la performance [23]. Le développement spectaculaire du *streaming* au cours des dernières années, tout comme la crise actuelle du coronavirus, ont pu contribuer à accroître le soin apporté à la projection du code et à en faire une composante essentielle de la préparation d'une performance. Thor Magnusson analyse, à propos de ce type d'art multimédia et sur un plan temporel plus large, la formation d'une nouvelle *« culture aurale »* [24], faisant du texte un élément transitoire parmi d'autres d'un art multimédia [12]. Si le

23 . Arts mécaniques, chorégraphiques, visuels, musicaux, parmi d'autres. Le festival Algomech (`https://algomech.com/2019/`) organisé le 19 mai 2019 à Sheffield en témoigne.

24 . *« We are witnessing the formation of a new type of oral culture (or rather aural culture), with improvisation and live coding on new instruments, one that is established through electronic media of audiovisual documentation »*.

texte reste souvent le point focal de la performance, nous remarquons que sa lisibilité n'est plus considéré comme une fin en soi mais comme l'un des éléments, parmi d'autres, qui participent à définir la performance.

Il nous semble que le texte algorithmique brut de la musique en *plain texte* établit le constat d'un échec partiel à communiquer et à témoigner de l'action musicienne. Bien qu'il soit une représentation du mouvement et de l'action, le texte n'est pas systématiquement considéré comme un document ou comme un texte lisible *en soi*. Représenter complètement la performance et la rendre lisible et intelligible pour le public est une tâche d'une extrême complexité, et peut-être même une aporie. Complémenter une performance *live codée* de visuels dynamiques constitue l'une des réponses possible à la difficulté de partager pleinement l'expérience du contact avec le système de création.

3.2.2. Intégration à des réseaux logiciels et insuffisance du texte

Faire sens du code utilisé par un *live coder* peut également s'avérer difficile pour la raison que ce dernier constitue autour de l'interface de *live coding* un écosystème d'instruments complexe et idiosyncratique, entièrement personnalisé. Les banques d'échantillons sonores ou les interconnexions logicielles sont autant d'éléments incommunicables et sans véritable représentation textuelle. Les extensions du langage, les raccourcis syntaxiques ou les éléments préprogrammés sont une autre catégorie d'éléments qui contribuent à séparer un état *standard* du langage utilisé de certains dialectes plus personalisés.

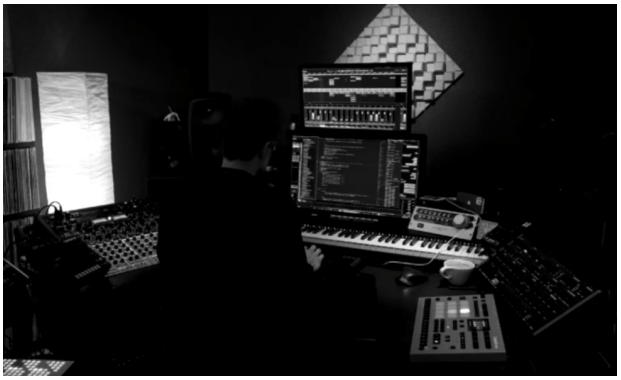

Figure 8. *Deru* (Benjamin Wynn, Los Angeles) utilisant *Tidal Cycles* lors d'une performance en *live-stream* avec *Speedy J* (Jochem George Paap, Rotterdam). Source : **SHS 33 Deru x Speedy J live**.

Une performance de *live coding* est techniquement basée sur le couplage mou entre plusieurs systèmes interconnectés, qui lient l'éditeur de texte à l'ordinateur du musicien, à un moteur audio ou à des instruments tiers. Ceux-ci s'avèrent parfois impossibles à présenter en *plain-texte* au public, ne possèdant aucune représentation sous la forme de code source. La librairie de *live-coding* dont le code est projeté ne constitue à ce titre que l'interface la plus abstraite d'un dispositif souvent invisible, dissimulé en *arrière plan* de la performance. *Tidal Cycles* fonctionne, à titre d'exemple, sur un réseau logiciel dépendant, *a minima* de *SuperCollider* et de ses capacités *I/O* :

Niveau	Interface	Input	Output
Contrôle	Tidal Cycles	Texte MIDI OSC	OSC
Interprétation	GHC	Source	OSC
Moteur d'exécution	SCLang	OSC	OSC
Moteur de synthèse	SCSynth	OSC	Audio

Table 1. Fonctionnement minimal de *Tidal Cycles* : couplage entre *Tidal* (Haskell), *SuperDirt* (moteur d'exécution) et SCSynth (moteur de synthèse).

Bien qu'une partie du texte soit projetée, ce dernier ne constitue le plus souvent qu'une représentation distante du fonctionnement réel du système. Les dépendances du texte algorithmique au système informatique du musicien et à son état rendent caduques ou complexifient considérablement les possibilités d'échange ou de partage de *patchs*, d'instruments ou de librairies entre musiciens. Le support du fichier texte, paradoxalement, n'est que rarement employé pour la sauvegarde ou pour le partage des informations et des données qui composent une pièce. La musique en *plain-texte*, dans sa forme la plus courante, ne peut donc pas être assimilée à une forme de notation traditionnelle sur support fixe. Le texte demeure au stade de trace, d'image transitoire du flux de l'improvisation, et ses altérations successives ne sont pas sauvegardées.

Notre notion de *plain-texte* trouve ici l'une de ses limites : une performance de *live coding* n'est que rarement définie *dans* le texte et *par* le texte seulement. Il est toutefois toujours possible d'utiliser cette notion comme nuancier pour définir plusieurs degrés d'inscription de la musique dans le texte. Un certain type de performance ne l'utilisera que comme simple interface de contrôle, déléguant toute définition de synthèse ou toute opération audio à un moteur externe. Une autre, plus explicite, se servira du texte et des capacités d'un langage de programmation pour définir les instruments, les graphes audios et les contrôles au cours de la performance, tout en les séquençant. Certains *live coders*, à l'instar de sean Cotterill, connu sous le nom de scène de *co34pt*, sont ainsi réputés pour leur ascétisme, n'utilisant souvent qu'une plateforme généraliste (ici *SuperCollider*) pour leurs performances.

La notion de *plain-texte* permet donc, on le voit, de réfléchir au degré d'inscription de la musique improvisée par le musicien. Elle permet tout autant de réfléchir à la nécessité de trouver, pour la sauvegarde de cet art, des formats d'échange et de partage permettant de répéter ou de garder trace du moment transitoire de l'édition du texte.

4. CONCLUSIONS ET PERSPECTIVES

Le texte brut ne peut pas être perçu comme un support neutre. Le *live coding* investit ce support d'une charge symbolique héritée d'autres cultures informatiques. Le choix de pratiquer la musique en *plain-texte* témoigne d'une volonté de la part du musicien de s'investir pleinement dans la maîtrise de son outil, de trouver un espace d'expression artistique en synergie avec le dispositif technique. Envisager le langage de programmation ou l'ordinateur comme instrument musical constitue un défi, et répond au désir de réinvestir un outil dont la modalité de fonctionnement échappe souvent à son utilisateur. La manipulation du texte réintroduit une dimension humaine d'écriture et d'inscription dans un mécanisme algorithmique déterministe, permettant de *« remettre du rythme dans l'algorithme »* ou de *« voler l'intelligence à l'intelligence artificielle »* [25] .

Quels apport peut-on tirer de la définition de la notion de *plain-texte* ? Comment la mobiliser au profit du perfectionnement des outils servant à la pratique ? Nous espérons, au cours de futurs travaux, proposer des solutions à même de faciliter l'apprentissage, la diffusion et l'archivage des oeuvres issues des pratiques musicales en *plain texte*. Il nous faudra, pour ce faire, tenir compte des spécificités du texte brut, et faire de ce matériau lui-même l'un des supports pour la documentation et l'échange entre utilisateurs autour de ces techniques.

4.1. Représenter plusieurs niveaux du plain-texte

La notion de *plain-texte* nous invite à distinguer différents degrés de notation algorithmique d'une performance, allant d'une notation au niveau du signal (*DSP*), à une notation/action de la performance de plus haut niveau. Il s'agit, en cela, d'une forme de notation musicale stratifiée. Le niveau de contrôle que la manipulation directe du code offre au musicien nous encourage, suivant Heinrich Taube, à imaginer le texte du *live coder* comme un *« métaniveau »*, niveau de l'écriture musicale dépassant celui de la seule partition : *« Si la partition représente la composition, alors le métaniveau représente la composition de la composition. Une représentation du métaniveau musical doit représenter l'activité, le processus de la composition musicale et non son artefact, la partition »* [28]. La notion de *plain-texte* peut servir à articuler plus clairement au sein des interfaces et des langages l'existence de plusieurs niveaux imbriqués de l'écriture algorithmique et à les présenter plus clairement à l'interprète et à son public.

4.2. Noter l'improvisation et le mouvement

Le texte manipulé par le musicien est un matériau en mouvement, toujours promis à une altération future ou à une redéfinition. Pour cette raison au moins, la captation vidéo constitue l'un des moyens les plus efficaces de *garder trace* du mouvement. La notion de *plain-texte* rend sensible le besoin d'établir une technique de notation capable de sauvegarder en *plain-texte* les différents états d'une improvisation afin de pouvoir retracer le fil d'une improvisation et la conserver. La mise en place d'un système de versionnage permettant de garder trace des étapes de l'improvisation pourrait constituer un atout majeur pour l'archivage ou la pédagogie de cette pratique.

4.3. Échange, documentation et archivage

La documentation existante concernant le *live coding* (guides, documents techniques) s'étale aujourd'hui sur des sites et des supports variés (*MediaWiki*, *GitHub*, *YouTube*, articles académiques, blogs), parfois soumis à une forme de fragmentation ou de péremption (*chats* informels, *mailing lists*) [26] . L'essentiel de cette documentation est le fait des développeurs et des créateurs eux-mêmes, suivis dans une moindre mesure par des groupes d'utilisateurs.

Nous pensons que la création d'outils spécifiques permettant aux *live-coders* d'échanger directement des fichiers bruts, des échantillons sonores et des banques de code téléchargées en local constituerait une alternative centralisée à ce phénomène de fragmentation et de dissémination des ressources. La documentation intégrée aux éditeurs de *Sonic Pi*, *FoxDot* ou *SuperCollider* constitue un modèle qu'il s'agit à présent d'étendre à d'autres plateformes (*par ex. Tidal Cycles*), tout en conservant les qualités intrinsèques du *plain-texte*.

4.4. Remerciements

La rédaction de cet article s'inscrit dans les stades préliminaires d'une recherche doctorale financée (bourse de thèse) par l'école doctorale 3LA (ED 484), et encadrée par le laboratoire ECCLA de l'université Jean-Monnet de Saint-Étienne. Nous tenons à les remercier de leur soutien et de leur confiance.

Nous adressons nos plus chalereux remerciements à Laurent Pottier et Alain Bonardi, directeurs de ce travail de recherche, pour le soutien et les conseils qu'ils ont apporté en vue de la rédaction de cet article.

5. REFERENCES

[1] Alexandro, Ludovico, « I think in text ». *Neural Magazine*, numéro 27, Juin 2007, *in* McLean Alex, « Neural interview on live coding », article de blog : https ://slab.org/neural-interview-on-live-codin/.

[2] Aaron, Samuel, et Alan F. Blackwell. « From Sonic Pi to Overtone : Creative Musical Experiences with Domain-Specific and Functional Languages ». *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling Design - FARM '13*, *ACM Press, 2013*, p. 35. DOI.org (Crossref), doi :10.1145/2505341.2505346.

25 . Slogans TOPLAP : `https://toplap.org/wiki/ToplapSlogans`

26 . Pour un bref tour d'horizon, consulter la notice *Awesome Live-Coding* sur GitHub : `https://github.com/toplap/awesome-livecoding`

[3] Bacot, Baptiste, et Clément Canonne. « Musique et hacking : de l'éthique aux pratiques ». *Volume !*, numéro 16 : 1, décembre 2019, p. 7-14. DOI.org (Crossref), doi :10.4000/volume.7211.

[4] Blackwell, Alan, et Nick Collins. « The Programming Language as a Musical Instrument ». 2005, *Proceedings of 17th Psychology of Programming Interest Group 2005*, University of Sussex.

[5] Brown, Andrew R, et Andrew C. Sorensen. « aa-cell in practice : An approach to musical live coding ». *International Computer Music Conference*, 2007, Copenhague, p. 292-99.

[6] Collins, Nick, et al. « Live Coding in Laptop Performance ». *Organised Sound*, vol. 8, numéro 3, décembre 2003, p. 321-30. DOI.org (Crossref), doi :10.1017/S135577180300030X.

[7] Collins, Nick. « Live Coding of Consequence ». *Leonardo Music Journal*, vol. 44, numéro 3, juin 2011, p. 207-11.

[8] Chesire, Tom, « Hacking meets clubbing with the 'Algorave' », *Wired Magazine*, 29 août 2013, https ://www.wired.co.uk/article/algorave.

[9] Dahlstedt, Palle. « Action and Perception : Embodying algorithms and the extended mind ». *The Oxford Handbook of Algorithmic Music*, Oxford University Press, 2018, p. 41-66.

[10] Kirkbride, Ryan. « FoxDot Live Coding with Python and SuperCollider ». *Proceedings of the International Conference on Live Interfaces*, University of Sussex, 2016.

[11] Magnusson, Thor. « Herding Cats : Observing live coding in the wild », *Computer Music Journal*, 38 (1), pp. 8-16, ISSN 0148-9267.

[12] Magnusson, Thor. Sonic writing : technologies of material, symbolic and signal inscriptions. *Bloomsbury Academic*, 2019.

[13] McCartney, James. « Rethinking the Computer Music Language : SuperCollider ». *Computer Music Journal*, vol. 26, n 4, décembre 2002, p. 61-68. DOI.org (Crossref), doi :10.1162/014892602320991383.

[14] McLean, Alex, et Geraint Wiggins. « Tidal–pattern language for the live coding of music ». *Proceedings of the 7th sound and music computing conference*, Barcelone, 2010.

[15] McPherson, Andrew, et Koray Tahıroğlu. « Idiomatic patterns and aesthetic influence in computer music languages ». *Organised Sound*, vol. 25, numéro 1, Cambridge University Press, 2020, p. 53-63.

[16] Mori, Giovanni. *Live coding ? What does it mean ? an ethnographical survey on an innovative improvitional approach*. éditions *Aracne*, 2020.

[17] Nilson, Click. « Live Coding Practice ». *Proceedings of the 7th International Conference on New Interfaces for Musical Expression - NIME '07*, ACM Press, 2007, p. 112. DOI.org (Crossref), doi :10.1145/1279740.1279760.

[18] Raymond, Eric. *The Art of UNIX Programming*. 1ère édition, Addison-Wesley, 2003.

[19] Roberts, C. Realtime Annotations Visualizations in Live Coding Performance. *Proceedings of the 2018 LIVE Programming Workshop*, 2018.

[20] Roberts, Charlie, et Graham Wakefield. *Tensions and Techniques in Live Coding Performance*. Édité par Roger T. Dean et Alex McLean, vol. 1, *Oxford University Press*, 2018. DOI.org (Crossref), doi :10.1093/oxfordhb/9780190226992.013.20.

[21] Rohrhuber, Julian et al. « Algorithms Today : Notes on Language Design for Just in Time Programming. »*International Computer Music Conference* (2005).

[22] Rohrhuber, Julian. « Algorithmic Music and the Philosophy of Time ». *The Oxford Handbook of Algorithmic Music*, *Oxford University Press*, 2018. Zenodo, doi :10.5281/zenodo.2596675.

[23] Sayer, Tim. « Cognitive Load and Live Coding : A Comparison with Improvisation Using Traditional Instruments ». *International Journal of Performance Arts and Digital Media*, vol. 12, numéro 2, juillet 2016, p. 129-38. DOI.org (Crossref), doi :10.1080/14794713.2016.1227603.

[24] Sorensen, Andrew. « Extempore. The design, implementation and application of a cyber-physical programming language ». Thèse de doctorat, *Australian National University*, Juin 2018.

[25] De Souza, Jonathan. *Music at Hand : Instruments, Bodies, and Cognition*. *Oxford University Press*, 2017.

[26] Stevenson Angus et Maurice White, « Plain », *Concise Oxford English Dictionary*, 12ème édition, 2011.

[27] Ogborn, David, et al. « Extramuros : making music in a browser-based, language-neutral collaborative live coding environment ». *Proceedings of the First International Conference on Live Coding*, 2015, p. 163-69.

[28] Taube, Heinrich. *Notes from the Metalevel : An Introduction to Computer Composition*. 0 éd., *Routledge*, 2013. DOI.org (Crossref), doi :10.4324/9781315078182.

[29] Himanen, Pekka. *The Hacker Ethic, and the Spirit of the Information Age*. *Random House*, 2001.

[30] Ge, Wang. « The Chuck Audio Programming Language, « A Strongly-Timed and On-The-Fly Environ/-Mentality » ». Thèse de doctorat, *Princeton University*, 2008.

[31] Ward, Adrian, Rohrhuber, Julian, Olofsson, Fredrik, McLean, Alex, Griffiths, Dave, Collins, Nick and Alexander, Amy. « Live algorithm programming and a temporary organisation for its promotion ». *Proceedings of the README Software Art Conference*, 2004.