

## **Read\_Me, Run\_Me, Execute\_Me : Malaise dans le logiciel**

**ou**

## **C'est la performativité du code, idiot<sup>1</sup> !**

> Inke Arns

Le titre de cet article fait référence à *Das Unbehagen in der Kultur*, ou, dans sa traduction française, *Malaise dans la civilisation*, un texte de Sigmund Freud datant de 1929 dans lequel il décrit la civilisation comme une fine pellicule qui recouvre et réprime les structures instinctives. Il n'est pas dans mon intention d'entrer ici dans une discussion détaillée de la théorie freudienne, mais ce qu'il est intéressant de noter dans le contexte de la culture informatique est le fait que le software art semble mettre au jour des structures logicielles et des architectures de code habituellement masquées par des surfaces ou des interfaces graphiques utilisateur (IGU). Le software art révèle ainsi ce qui demeure généralement invisible : le code source et sa performativité.

Depuis quelques années, le terme d'art génératif est à la mode. On le rencontre dans des contextes très divers, aussi bien dans les discours universitaires que dans les festivals d'art médiatique, en dessin industriel ou en architecture. Très souvent, le terme est utilisé, sinon comme un exact synonyme, du moins comme un équivalent du terme " software art ". D'une certaine façon, l'art génératif et le software art sont en effet liés, mais la nature exacte de cette relation n'est jamais explicitée. Un des objectifs de cet article est de mettre en lumière *ce lien entre art génératif et software art*. L'autre objectif est de proposer la notion de *performativité du code* comme étant l'une des raisons pour lesquelles les artistes contemporains s'intéressent au logiciel en tant que matériau artistique. Dans ce contexte, la performativité du code renvoie à sa capacité à agir et à intervenir au sens où l'entend la théorie de l'acte de parole.

### **I - Art génératif ≠ software art**

Selon Philip Galanter (2003), l'art génératif définit " *toute pratique artistique dans laquelle l'artiste utilise un système, tel qu'un ensemble de règles de langage naturel, un programme informatique, une machine ou toute autre invention procédurale, qui sont alors mis en mouvement avec un certain degré d'autonomie, contribuant en partie ou en totalité à la création d'une œuvre d'art complète*<sup>2</sup> ". L'art génératif décrit ainsi des processus définis par des règles, des processus qui sont automatisés jusqu'à un certain point par l'utilisation d'une machine ou d'un ordinateur, ou par l'utilisation d'instructions mathématiques ou pragmatiques. Une fois mis en mouvement, selon les règles ou instructions prédéfinies, ces processus " auto-organisés " fonctionnent indépendamment de leurs auteurs ou artistes-programmeurs. Suivant le contexte technique dans lequel se déroule le processus, le résultat est "imprévisible ". Il procède donc moins d'une action individuelle ou d'une "paternité" artistique que des conditions respectives de

---

<sup>1</sup> Cet article est tiré d'une conférence prononcée à l'occasion du colloque international " Programmation orientée art - Décodage et critique ", tenu à la Sorbonne les 19 et 20 mars 2004. [publié en anglais dans : Olga Goriunova, Alexei Shulgin (Sous la direction de), *Read\_Me. Software Art & Cultures*, Aarhus, 2004, pp. 176-193.]

<sup>2</sup> Philip Galanter: "What is Generative Art ? Complexity Theory as a Context for Art Theory", *Generative Art Proceedings*, Milan, 2003, p.4 ; <http://www.philipgalanter.com/pages/acad/media/ga2003%20proceedings%20paper.pdf>. La liste de diffusion eu-gene est consacrée au débat sur l'art génératif, voir <http://www.generative.net/>.

production, ou, si l'on veut, de l'écologie technique d'un système donné<sup>3</sup>. Comme les définitions qu'en donnent différents auteurs, la définition que propose Galanter de l'art génératif est une définition " globale ", totale et complète, qui l'amène à la conclusion, qui peut paraître surprenante, que " *l'art génératif est aussi ancien que l'art lui-même*<sup>4</sup> ".

Mais revenons-en à la caractéristique principale : l'élément le plus remarquable dans toutes ces tentatives de définition de l'art génératif - que ce soit dans la musique électronique ou la composition algorithmique, le dessin ou l'animation sur ordinateur, la scène Démo ou la culture VJ (video-jockey), le dessin industriel ou l'architecture<sup>5</sup> -, c'est le recours à des processus génératifs visant à la *négation de l'intentionnalité*. L'art génératif ne s'intéresse aux processus génératifs (ainsi qu'au logiciel ou code) que dans la mesure où ils génèrent des événements "imprévisibles". En ce sens, et dans ce contexte, logiciel et code sont considérés comme des outils pragmatiques qui ne sont pas eux-mêmes questionnés. Et c'est précisément pour cette raison - le fait que l'art génératif se concentre sur la négation de l'intentionnalité sans s'attacher à questionner les outils employés - que la notion d'art génératif ne saurait - en tout cas dans la plupart des cas - être employée comme synonyme de software art.

Le software art décrit une activité artistique qui, dans le médium - ou plutôt le matériel - du logiciel, autorise une réflexion critique sur le logiciel (et sur son impact culturel). Le software art ne considère pas seulement le logiciel comme un outil pragmatique invisible générant certains résultats ou surfaces visibles, mais au contraire se concentre sur le code source lui-même - même si ce code n'est pas exposé ou mis en avant de manière explicite. D'après Florian Cramer, le software art révèle les implications esthétiques et politiques sous-jacentes présentes dans des commandes techniques apparemment neutres. Aussi le software art est-il susceptible d'intervenir à différents niveaux : au niveau du code source, au niveau d'algorithmes abstraits ou encore au niveau du résultat généré par un certain code programme. Il n'est donc pas surprenant qu'existe une grande variété de software art, depuis ce que l'on appelle des " œuvres code ", constituées pour l'essentiel de code ASCII (qui ne sont pas *exécutables*), jusqu'à des navigateurs Web expérimentaux (par exemple le *WebStalker* d'I/O/D créé en 1997) et des programmes entièrement exécutables (comme *Vexation 1*<sup>6</sup> d'Antoine Schmitt, ou *Auto-Illustrator*<sup>7</sup> d'Adrian Ward, tous deux réalisés en 2000). Dans l'art génératif, le logiciel n'est qu'un matériau parmi d'autres. Le software art, d'autre part, peut receler des éléments d'art génératif, mais il n'a pas forcément à être génératif au sens strictement technique (*cf.* les " œuvres code "). Les termes de software art et d'art génératif ne peuvent donc pas être considérés comme synonymes. Disons plutôt que ces deux notions fonctionnent sur des *registres différents*, ce que je vais tenter de démontrer par les exemples qui suivent.

Mon premier exemple sera le projet *insert\_coin*<sup>8</sup> de Dragan Espenschied et Alvar Freude. Dans le cadre de la préparation de leur diplôme, qu'ils présentèrent en 2000-2001 sous l'argument : " 2 personnes contrôlent 250 personnes ", les deux étudiants en art multimédia installèrent secrètement un serveur Web proxy à l'académie Merz de Stuttgart, serveur grâce auquel, *via* un script Perl, ils purent manipuler la totalité du trafic Internet des étudiants et enseignants connectés au réseau de l'académie. D'après Espenschied et Freude, l'objectif de ce projet était de procéder à une évaluation critique de la " *compétence et de la faculté critique des*

---

<sup>3</sup> Voir aussi Cox, G. : "Anti-thesis: the dialectics of generative art (as praxis) ", MPhil/PhD Transfer Report 2002, <http://www.anti-thesis.net/>. On trouve une définition similaire dans Adrian Ward : « *On appelle art génératif le travail sur les processus concourant à donner naissance à une œuvre d'art, généralement (mais pas uniquement) automatisés grâce à l'utilisation d'une machine ou d'un ordinateur, ou qui utilise des instructions mathématiques ou pragmatiques pour définir les règles suivant lesquelles ces œuvres d'art sont générées.* » (<http://www.generative.net>)

<sup>4</sup> Philip Galanter, *ibid.*, p. 1.

<sup>5</sup> Galanter, *ibid.*, p.2, appelle ces quatre domaines les quatre " *bouquets principaux* " de l'art génératif.

<sup>6</sup> <http://www.gratin.org/as/>

<sup>7</sup> <http://www.signwave.co.uk>

<sup>8</sup> [http://www.odem.org/insert\\_coin/](http://www.odem.org/insert_coin/)

*utilisateurs concernant l'usage quotidien du médium Internet*<sup>9</sup>. Le serveur proxy manipulé redirigeait les URL entrées par l'utilisateur vers d'autres adresses, modifiait le code HTML, transformait la teneur des dernières actualités des sites d'information grâce à une simple fonction chercher-remplacer (par exemple en modifiant le nom des politiciens cités) ainsi que le contenu des e-mails privés qui étaient lus par des interfaces Web comme *Hotmail* ou *Yahoo!*. Pendant quatre semaines, ce projet a pu manipuler l'accès Internet de l'académie entière - et ce sans que quiconque ne le remarque. Lorsque Espenschied et Freude révélèrent publiquement la manipulation, presque personne n'y prêta attention. Les deux étudiants publièrent même un manuel pratique permettant à chacun de désactiver très facilement le filtre responsable de la manipulation du contenu en ligne. Pourtant seule une minorité de leurs condisciples et professeurs prirent le temps d'opérer cet ajustement mineur permettant de retrouver une information non filtrée. Plusieurs mois après la fin de l'expérience, l'accès Internet de la plupart des ordinateurs de l'académie était toujours filtré.

Mon second exemple, *walser.php*, réalisé en 2002 par *textz.com/Project Gutenberg* (c'est-à-dire Sebastian Lütgert), a été qualifié de logiciel "politique" ou "littéraire"<sup>10</sup>. On pourrait le définir comme un logiciel militant anti-copyright, écrit en réponse à l'un des plus grands scandales littéraires survenus en Allemagne depuis la Seconde Guerre mondiale. Le nom de fichier *walser.php* n'est pas seulement une allusion ironique au fichier "walser.pdf", une version numérique pré-imprimée du roman controversé de Martin Walser, distribuée par la maison d'édition Suhrkamp comme pièce jointe à un e-mail - avant d'être plus tard, vu les circonstances défavorables, rappelée par l'éditeur (opération délicate, le rappel d'un document électronique...). *Walser.php* (ou plutôt *walser.pl*) de *textz.com* était et est toujours un script *Perl* qui, grâce à un interpréteur *Perl* adéquat, peut, à partir des 10 000 lignes de code source<sup>11</sup>, générer une version texte ASCII du roman de Walser, *Mort d'un critique*, lisible par un être humain. Si le code source écrit en *Perl* contient le roman lui-même sous une forme "invisible" lisible par une machine, et peut donc être distribué et modifié en tant que logiciel gratuit régi par la *GNU General Public License*, il ne peut être exécuté qu'après autorisation écrite des éditions Suhrkamp<sup>12</sup>.

Tandis que l'expérience d'Espenschied et Freude, consistant à filtrer et à censurer le contenu Internet, révèle le potentiel relativement illimité de contrôle que possèdent les logiciels, *walser.php* offre une solution pratique aux restrictions commerciales qui menacent la liberté d'information sur Internet sous la forme des Digital Rights Management Systems (DRM) [Systèmes de gestion des droits numériques]. Alors que *insert\_coin* réalise temporairement un scénario contre-utopique sous la forme d'un logiciel manipulé, *textz.com* développe avec son projet *walser.php* des "contre-mesures logicielles"<sup>13</sup> authentiquement utopiques.

Ces deux projets sont *génératifs* dans la meilleure acceptation du terme. Pourtant, ni *insert\_coin* ni *walser.php* ne correspondent aux définitions de "l'art génératif" qui prédominent dans le domaine du design. Philip Galanter, par exemple, que j'ai cité plus haut, définit l'art génératif comme un processus *contribuant en partie ou en totalité à la création d'une œuvre d'art complète*. De la même façon, Celestino Soddu, directeur du *Generative Design Lab* de l'université polytechnique de Milan et organisateur des conférences sur l'Art génératif<sup>14</sup>, définit ce dernier comme un outil processuel permettant à l'artiste ou au concepteur de "synthétiser [...] une série aléatoire et imprévisible d'événements, d'images, d'objets industriels, d'architectures, d'œuvres musicales, d'environnements et de communications"<sup>15</sup>.

---

<sup>9</sup> Cf. le texte présenté par Espenschied et Freude à l'occasion de l'Internationaler Medienkunstpreis 2001, [http://www.onlinedemonstration.org/insert\\_coin/imkp2001.html](http://www.onlinedemonstration.org/insert_coin/imkp2001.html)

<sup>10</sup> Cramer, F. : "walser.php", in : Goriunova, O. / Shulgin, A. (sous la direction de) : *Read\_Me 2.3. Reader*. Nifca, Helsinki, 2003, pp.76–78 ; ici : p.76.

<sup>11</sup> <http://textz.com/trash/walser.pl.txt>

<sup>12</sup> Cf. *textz.com* : "Suhrkamp calls back walser.pdf, textz.com releases walser.php", <http://textz.com/trash/readme.txt>

<sup>13</sup> Cramer, *ibid.*, p. 77

<sup>14</sup> Cf. <http://www.generativeart.com/>

<sup>15</sup> Soddu, C. : "Generative Art and Architecture", *Abstract*, non daté,

On comprend à la lecture de ces définitions que l'art génératif s'intéresse en premier lieu aux *résultats* produits par les processus génératifs. Dans ce contexte, le logiciel est considéré et employé comme un outil ou un appareil pragmatique-génératif en vue de la création de certains résultats - sans être lui-même questionné. Les processus génératifs produits ici par le logiciel servent avant tout à échapper à l'intentionnalité et à produire une diversité inattendue, arbitraire et inépuisable de formes. La *n\_Gen Design Machine* de Move Design, présentée en 2003 au festival *Read\_Me* d'Helsinki, comme le *net-art Generator*<sup>16</sup>, réalisé en 1999 par Cornelia Sollfrank et qui, par une simple pression sur un bouton, génère du Net Art, devraient être considérés comme des commentaires ironiques sur le " design génératif " mal compris<sup>17</sup>.

*insert\_coin* et *walser.php* vont au-delà de ces définitions de " l'art génératif " ou du " design génératif " dans la mesure où ces projets portent une attention beaucoup plus grande aux *processus codés* générant certains résultats ou surfaces. Cet intérêt porté aux processus codés ou, plus précisément, à la signification et aux implications du logiciel et des structures codées, les distingue nettement non seulement de l'art génératif mais aussi de nombreuses installations interactives des années 1990 qui manifestaient leur désintérêt envers le logiciel en dissimulant le code programme dans des *boîtes noires*. Au contraire, des projets comme *insert\_coin* et *walser.php* ont pour objectif de questionner le logiciel et le code en tant que culture - et de questionner la culture en tant que logiciel appliqué. Dans cette perspective, ils développent un "logiciel expérimental" (dans *insert\_coin* un serveur proxy et dans *walser.php* un script *Perl*) qui non seulement génère des surfaces arbitraires, mais explore de façon critique l'impact technologique, culturel ou social du logiciel. De plus, l'écriture d'un "logiciel expérimental" s'intéresse beaucoup plus à la *subjectivité artistique*, telle qu'elle s'exprime dans l'utilisation de différents langages privés, qu'à la démonstration d'une "créativité machinique" (quelle que soit la signification de ce terme) : "Un code peut relever du journal intime, il peut être obscur, poétique, ironique ou perturbateur, dépassé ou impossible, il peut stimuler et camoufler, il possède un rythme et un style, il peut être une attitude"<sup>18</sup>, telle est la vibrante définition qu'en donnent Florian Cramer et Ulrike Gabriel, tous deux membres du jury du festival de software art *Transmediale* en 2001.

J'ai pour ma part tenté d'établir une comparaison quelque peu polémique entre art génératif et software art. La voici :

### Art génératif

### Software art

S'attache avant tout à la surface (phénotexte) créée par un processus génératif ("problème de la boîte noire")

S'attache au processus génératif (déclenché par un " génotexte ") susceptible de générer des surfaces ou d'autres résultats.

Considère le logiciel comme un outil pragmatique/neutre servant à produire un certain résultat ; l'outil lui-même n'est pas questionné.

Aborde le logiciel comme une culture qu'il s'agit de questionner ; s'intéresse aux significations esthétiques et politiques sous-jacentes ; le logiciel peut être " expérimental " et " non pragmatique ".

Le logiciel en tant qu'outil pragmatique-génératif.

Le logiciel ou le code en tant qu'œuvre à part entière (éventuellement expérimentale)

Code efficace (" de beaux algorithmes\* ")

Le code comme excès, le code comme extravagance, pas nécessairement efficace.

---

<http://www.nyu.edu/studio/generative.html>

<sup>16</sup> Cornelia Sollfrank, net.art generator (1999), <http://soundwarez.org/generator/>

<sup>17</sup> Cf. Goriunova, O. / Shulgin, A. : " n\_Gen Design Machine ", in : Goriunova, O. / Shulgin, A. (Sous la direction de): *Read\_Me 2.3. Reader*, Nifca, Helsinki, 2003, pp. 66–67 ; ici : p.66.

<sup>18</sup> Cramer, F. / Gabriel, U. : " Software Art ", in : Broeckmann, A. / Jaschko, S. (Sous la direction de) : *DIY Media – Art and Digital Media : Software - Participation - Distribution*. Transmediale.01, Berlin, 2001, pp.29–33 ; ici : p.33.

Emploi de processus génératifs destinés à nier l'intentionnalité. “ *Les software artists [...] semblent concevoir les systèmes génératifs non comme une négation de l'intentionnalité, mais comme une recherche d'équilibre entre hasard et contrôle. [...] Loin d'être seulement un art destiné aux machines, le software art s'intéresse beaucoup à la subjectivité artistique, à la façon dont elle est reflétée et multipliée dans les systèmes génératifs*” (Cramer/Gabriel)

Fascination du génératif

S'intéresse à la “ performativité ” du code.

\* Cf. Donald Knuth, *The Art Of Computer Programming : Vol. 1, Fundamental Algorithms*, Reading, Mass., 1997.

\*\* Florian Cramer / Ulrike Gabriel, cités par Andreas Broeckmann, “On Software as Art”, in : *Sarai Reader 2003 : Shaping Technologies*, New Delhi, 2003, pp. 215-218 ; ici : p. 216.

Le concept de software art (ou de “ logiciel artistique ”) fut formulé pour la première fois et introduit comme catégorie en compétition lors du festival d'art médiatique *Transmediale* de Berlin en 2001.<sup>19</sup> <sup>20</sup> Le software art, que certains auteurs appellent logiciel “ expérimental<sup>21</sup> ” ou “ spéculatif<sup>22</sup>”, ou encore logiciel “ non pragmatique ” ou “ non rationnel<sup>23</sup>”, englobe, si nous nous

---

<sup>19</sup> Autres événements notables : le Kontrollfelder, organisé en 2001 à Dortmund par Andreas Broeckmann et Matthias Weiß, <http://www.hartware-projekte.de/programm/inhalt/kontroll.htm> ; le Read\_Me Festival,

conçu par Olga Goruïnova et Alexei Shulgin (Moscou 2002, Helsinki 2003, [http://www.m-cult.org/read\\_me/](http://www.m-cult.org/read_me/)) et les expositions Generator (GB 2002, organisée par Geoff Cox, <http://www.generative.net/generator.html>), CODEDOC (New York, sept. 2002, organisée par Christiane Paul, <http://artport.whitney.org/commissions/codedoc/>), I love you - computer\_viren\_hacker\_kultur (Francfort/Main, 31 jan-5 fév. 2003, [http://www.digitalcraft.org/index.php?artikel\\_id=269](http://www.digitalcraft.org/index.php?artikel_id=269)) et l'entrepôt d'art logiciel Runme, lancé en janvier 2003 (<http://runme.org>). On trouvera d'autres exemples d'art logiciel sur ces sites Web. L'année historiquement la plus significative sur le plan de l'art logiciel fut l'année 1970, durant laquelle se déroulèrent trois événements importants : l'exposition de Jack Burnham : Software – Information Technology : Its New Meaning for Art, tenue au Musée juif de New York ; l'exposition organisée par Kynaston McShine au MoMA de New York, sous le titre “ Information ” ; enfin la fondation du magazine *Radical Software* par Beryl Korot, Phyllis Gerhuny et Ira Schneider (<http://www.radicalsoftware.org/>).

<sup>20</sup> On trouvera l'un des premiers articles conceptuels concernant l'art et la programmation logiciels dans Cox, G. / McLean, A. / Ward, A. : “ The Aesthetics of Generative Code ” (2000), <http://generative.net/papers/aesthetics/>. On trouvera également une tentative de définition et d'exploration formelle des débuts de l'art logiciel utilisant des exemples littéraires et artistiques dans Cramer, F. : “ Concepts. Notations. Software. Art ”, 23 mars 2002, [http://userpage.fu-berlin.de/~cantsin/homepage/writings/software\\_art/concept\\_notations/concepts\\_notations\\_software\\_art.html](http://userpage.fu-berlin.de/~cantsin/homepage/writings/software_art/concept_notations/concepts_notations_software_art.html)

<sup>21</sup> Baumgärtel, T. : “ Experimentelle Software. Zu einigen neueren Computerprogrammen von Künstlern ”, Telepolis, 28 oct. 2001, <http://www.heise.de/tp/deutsch/inhalt/sa/9908/1.html>

<sup>22</sup> Matthew Fuller distingue logiciel “ critique ”, “ social ” et “ spéculatif ”. Cf. Fuller, M. : “ Behind the Blip : Software as Culture ”, *Nettime*, 7 janv. 2002, <http://amsterdam.nettime.org/Lists-Archives/nettime-l-0201/msg00025.html>

<sup>23</sup> Olga Goruïnova et Alexei Shulgin définissent le “ logiciel artistique ” comme “ non pragmatique ” et “ non rationnel ” : “ Si les programmes conventionnels sont des instruments servant des objectifs purement pragmatiques, le résultat du travail des programmes artistiques se situe souvent en dehors de tout pragmatisme et de toute rationalité ” (Goriunova, O. / Shulgin, A. :

en tenons à la définition proposée par le festival *Transmediale*, des projets dont le matériau artistique principal est le code programme, et qui portent une vision critique sur l'acceptation culturelle du logiciel. Le code programme n'est pas considéré comme un instrument pragmatique servant à accomplir telle ou telle tâche, mais comme un matériau générateur de processus mécaniques et sociaux. Sous cet aspect, le software art peut aussi bien constituer le résultat d'une pratique créative autonome et formelle que faire référence de manière critique à un logiciel existant, ainsi qu'à l'impact technologique, culturel ou social du logiciel.<sup>24</sup>

Il est intéressant de noter que la différence entre software art et art génératif rappelle celle existant entre les formes contemporaines du software art et les premières créations d'art numérique des années 1960 (et je fais référence ici au "logiciel expérimental" de Tilman Baumgärtel de 2001). Cette différence peut être énoncée de la façon suivante : les œuvres appartenant au domaine du software art ou du logiciel expérimental "*ne sont pas de l'art créé avec l'aide de l'ordinateur, mais de l'art qui survient dans l'ordinateur ; le logiciel n'est pas programmé par les artistes pour produire des œuvres autonomes, c'est le logiciel lui-même qui est l'œuvre d'art. L'essentiel ici n'est pas le résultat, mais le processus déclenché dans l'ordinateur par le code programme*"<sup>25</sup>. Bien que le computer art des années 1960 fût proche de l'art conceptuel en ce qu'il privilégiait le concept plus que la réalisation elle-même, il n'a cependant pas poussé cette idée à son terme : avec ses œuvres réalisées sur papier par des traceurs et imprimantes à aiguille, il mettait l'accent sur le produit fini, non sur le programme ou le processus ayant généré l'œuvre<sup>26</sup>. Dans les projets de software art actuels, ce rapport est inversé, puisqu'on s'y intéresse "*exclusivement au processus généré par ces programmes. Alors que le computer art des années 1960 et 70 considérait les processus internes à l'ordinateur comme une simple méthode pour générer un résultat externe, et non comme une œuvre en tant que telle, et traitait l'ordinateur comme une sorte de boîte noire, dissimulant ainsi les opérations et procédures qui s'y déroulaient, les projets actuels de software art entendent se focaliser sur ces processus, les rendre visibles et les soumettre à discussion*"<sup>27</sup>.

## II. Performativité du code contre fascination du génératif

L'intérêt actuel manifesté à l'égard du logiciel ne résulte pas uniquement à mes yeux de la fascination exercée par l'aspect génératif du logiciel, c'est-à-dire sur sa capacité à créer et à se reproduire au sens purement technique. Ce qui intéresse beaucoup plus les auteurs de ces projets, c'est ce que j'appelle la *performativité* du code. Par *performativité du code*, j'entends sa capacité à agir et à fonctionner au sens de la théorie des actes de langage.

Je pense ici à une série de conférences données par John Langshaw Austin à l'université Harvard en 1955. Dans ces conférences, intitulées *How to Do Things With Words*, Austin formula l'idée révolutionnaire selon laquelle le langage n'a pas seulement une fonction descriptive, référentielle ou constatative, mais qu'il possède également une dimension performative. Austin

---

"Artistic Software for Dummies and, by the way, Thoughts About the New World Order", *Nettime*, 26 mai 2002, <http://amsterdam.nettime.org/Lists-Archives/nettime-l-0205/msg00169.html>

<sup>24</sup> Cf. [http://www.transmediale.de/04/pdf/tm04clubtm04\\_formular\\_ausschreibung.pdf](http://www.transmediale.de/04/pdf/tm04clubtm04_formular_ausschreibung.pdf). Sur l'art logiciel, cf. le débat tenu lors du *transmediale.03* (Künstlerhaus Bethanien, Berlin, 4 fév. 2003), <http://www.softwareart.net/>, ainsi que Goriunova, O. / Shulgin, A. (Sous la direction de) : *Read\_Me 2.3 Reader - about software art*. Nifca, Helsinki, 2003. [http://www.m-cult.org/read\\_me](http://www.m-cult.org/read_me).

<sup>25</sup> Baumgärtel, *ibid.* [c'est moi qui souligne].

<sup>26</sup> Les œuvres de ceux qu'on appelle les "Algoristes", un groupe fondé par Roman Verostko, sont très représentatives de cette notion. Cf. Verostko, R. : "Epigenetic Painting : Software As Genotype, A New Dimension of Art" (1988), <http://www.verostko.com/epigenet.html> ; Verostko, R. : "Epigenetic Art Revisited : Software as Genotype", in : Schöpf, C. / Stocker, G. (Sous la direction de) : *Ars Electronica 2003 : Code - The Language of Our Time*. Cantz, Ostfildern, 2003, pp.156–167. Dans cet ouvrage, on trouve des formulations telles que : "*Le caractère essentiel de chaque œuvre découle de la "procédure génératrice de forme" ou "algorithme" qui agit comme un génotype. C'est pourquoi on peut dire que l'œuvre finale est une épiphanie, une manifestation de son générateur, le code. Pour moi, chaque œuvre célèbre son code.*"

<sup>27</sup> Baumgärtel, *ibid.*

distingue trois actes de langage différents : l'acte locutoire<sup>28</sup>, l'acte illocutoire<sup>29</sup> et l'acte perlocutoire<sup>30</sup>. Seuls les actes de langage illocutoires sont performatifs - c'est-à-dire qu'ils créent ou font ce qu'ils décrivent, pourvu qu'ils se situent au sein d'une matrice qui soit simultanément sociale et sémiotique. Ce qui souligne l'importance du *contexte* d'une énonciation performative. L'énonciation illocutoire, ou performative, peut réussir ou échouer en fonction du contexte, approprié ou non, dans lequel elle intervient.

Ainsi, si je parle de performativité du code, j'affirme que cette performativité ne doit pas se comprendre comme une performativité purement technique, c'est-à-dire qui se déroulerait exclusivement dans un système technique clos, mais comme quelque chose qui touche aux domaines de l'esthétique, du politique et du social. Le code programme se caractérise par le fait qu'ici, "dire" coïncide avec "faire". Le code en tant qu'acte de parole efficace n'est pas la description ou la représentation de quelque chose ; au contraire il affecte de manière directe et met littéralement en mouvement - voire même " tue " - un processus<sup>31</sup>. Cette " performativité codée<sup>32</sup> " a des conséquences politiques immédiates sur les espaces réels et virtuels (dont l'internet) dans lesquels nous vivons et évoluons de plus en plus : cela signifie en dernier ressort que cette performativité codée *mobilise* ou *immobilise* ses utilisateurs. Le code devient ainsi Loi ou, comme le disait Lawrence Lessig dès 1999 : " *Le code est [déjà] la Loi*<sup>33</sup>. " C'est la raison pour laquelle le software art s'intéresse plus à la "performance " qu'à la " compétence " (deux termes forgés par Noam Chomsky), plus à la *parole*\* qu'à la *langue*\*<sup>34</sup> (selon la fameuse opposition établie par Ferdinand de Saussure). Dans notre contexte, *performance* et *parole*\* renvoient aux actualisations, réalisations et répercussions concrètes qu'un certain code programme a sur, disons, les systèmes sociaux, et pas seulement à ce qu'il fait ou génère dans le contexte de systèmes abstraits-techniques. Dans les projets *insert\_coin* et *walser.php*, le génératif est profondément politique - et ce parce que la transformation secrète de textes existants (dans le cas d'*insert\_coin*) ou l'extraction d'un script Perl d'un texte protégé par les lois sur le copyright (dans le cas de *walser.php*) sont des actes qui peuvent être questionnés et critiqués *non pas* dans le contexte d'un système *technique*, mais dans le contexte de *systèmes politiques et sociaux* qui dépendent de plus en plus étroitement de ces structures techniques.

" *L'une des applications sans doute les plus radicales de l'idée de code informatique en tant que matériau artistique*<sup>35</sup> " est constituée par ce que l'on appelle les " *Codeworks*<sup>36</sup> " (œuvres code) et l'utilisation artistique qu'ils font du code programme. Les " œuvres code " consistent presque exclusivement en textes envoyés à des listes de diffusion comme *Nettime* ou *7-11* sous forme de simples e-mails. Les " œuvres code " utilisent le code d'instructions ASCII et son esthétique - sans

---

<sup>28</sup> Locutoire : acte de langage porteur d'une signification.

<sup>29</sup> Illocutoire : énoncé signifiant prononcé avec une certaine force conventionnelle qui produit non conventionnellement un certain effet.

<sup>30</sup> Perlocutoire : énoncé signifiant véhiculant une certaine force conventionnelle ou performative.

<sup>31</sup> Cf. Arns, I. : " Texte, die (sich) bewegen : zur Performativität von Programmiercodes in Netzkunst und Software Art ", in : Arns, I. / Goller, M. / Strätling, S. / Witte, G. (Sous la direction de) : *Kinetographien*, Aisthesis, Bielefeld, 2004 [à paraître].

<sup>32</sup> Grether, R. : " The Performing Arts in a New Era ", *Rohrpost*, 26 juillet 2001, <http://coredump.buug.de/pipermail/rohrpost/2001-July/000353.html>

<sup>33</sup> Lessig, L. : *Code and other Laws of Cyberspace*, Basic Books, New York, 1999.

\* En français dans le texte.

<sup>34</sup> C'est à la grammaire à transformation générative de Noam Chomsky que l'on doit la distinction entre compétence et performance (voir Chomsky, N. : *Aspects of the Theory of Syntax*, Cambridge, MA., 1965) ; la distinction entre langue et parole est attribuée à Ferdinand de Saussure (voir De Saussure, F. : *Cours de linguistique générale*, Paris, 1916 (réédit. 1967).

<sup>35</sup> Cramer, F. : " Exe.cut[up]able statements : Das Drängen des Codes an die Nutzeroberflächen ", in : Schöpf, C. / Stocker, G. (Sous la direction de) : *Ars Electronica 2003 : Code - The Language of Our Time*, Cantz, Ostfildern, 2003.

<sup>36</sup> Voir à ce sujet Sondheim, A. : " Codework ", in : *American Book Review*, vol. 22, n° 6 (septembre/octobre 2001), <http://www.litline.org/ABR/PDF/Volume22/sondheim.pdf>

pour autant avoir recours aux surfaces et interfaces utilisateur graphiques habituellement créées par ce code. Les œuvres de Jodi, de Netochka Nezvanova (alias Antiorp) et de Mez<sup>37</sup> rappellent ainsi l'existence d'un " monde invisible de processus<sup>38</sup>", comme le formule Graham Harwood. Techniquement parlant, ces " œuvres code " se situent à l'extrémité opposée d'un spectre imaginaire de générativité. Pourtant, le statut de ce langage ou de ces fragments de quasi-langage demeure ambigu : dans la perception du destinataire, ils oscillent entre l'exécutabilité supposée, et donc la fonctionnalité, et la non-exécutabilité - c'est-à-dire la dysfonctionnalité - du code ; en d'autres termes, ils se situent entre information signifiante et bruit dépourvu de sens. On peut observer très clairement ce phénomène dans l'e-mail *walkmonster\_start ()* que Jodi a envoyé à la liste de diffusion de *Nettime* le 22 octobre 2001. Alors que le texte contenu dans ce courrier ressemble à un code programme exécutable, la question reste ouverte pour le lecteur profane de savoir s'il peut être compilé ailleurs dans l'ordinateur, donc être transformé en algorithmes lisibles par la machine, et donc, en dernier ressort, être exécutable.

Ce qui joue un rôle majeur ici, plus que l'exécution technique effective, c'est la compréhension du fait que les fragments de code utilisés dans les œuvres code peuvent *potentiellement* être exécutés, et donc devenir performatifs. Toutefois, dans " The Aesthetics of Generative Code ", Geoff Cox, Alex McLean et Adrian Ward affirment que " *la valeur esthétique du code réside dans son exécution, et pas seulement dans sa forme écrite*<sup>39</sup> ". Si je peux être d'accord avec cette affirmation en ce qui concerne des projets comme *insert\_coin* et *walser.php* - parce que leur dynamisme critique (voire même poétique) réside précisément dans leur exécution technique - il faudrait étendre cette définition à la structure même des " œuvres code ". La valeur esthétique et poétique de ces " œuvres code " ne réside pas seulement dans leur forme textuelle, mais dans le fait - et la connaissance du fait - qu'ils sont *potentiellement* exécutables. J'aimerais élargir la notion du génératif, au sens où le code n'est pas seulement exécutable dans des environnements techniques, mais qu'il peut devenir extrêmement productif comme " logiciel imaginaire " chez le lecteur.

Contrairement à l'art génératif, le software art attire notre attention sur le fait que notre environnement (médiatique) repose de plus en plus sur des structures programmées. Ainsi, les " œuvres code " se servent du support " pauvre " du texte qui, en même temps, semble performatif, ou exécutable dans le contexte de la ligne de commande. En utilisant précisément cette ambivalence ou cette oscillation entre simplicité et totalité d'exécution, les œuvres code, et plus généralement le software art dans son ensemble, révèlent la dimension potentiellement totalitaire du code programme algorithmique, " *le monde invisible des processus* ".

---

<sup>37</sup> Pour d'autres exemples, voir le " <nettime> unstable digest " de Florian Cramer à l'adresse <http://www.nettime.org/archives.php>

<sup>38</sup> Harwood, G. : " Speculative Software ", in : Broeckmann A. / Jaschko, S. (Sous la direction de) : *DIY Media - Art and Digital Media : Software - Participation - Distribution*. Transmediale.01. Berlin, 2001, pp.47–49 ; ici : p.47.

<sup>39</sup> Cox, G. / McLean, A. / Ward, A. : " The Aesthetics of Generative Code "(2000), <http://generative.net/papers/aesthetics/>