# PPH: Le code est-il de l'art ?

Problématique

**Quelles sont les possibilités de faire de l'art avec du code ?**

Florian Rascoussier

INSA | INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

## Projet Personnel en Humanité

**Institut National des Sciences Appliquées de Lyon (INSA Lyon)**

# Rapport de soutenance

Semestre d'été 2022

Problématique

## Quelles sont les possibilités de faire de l'art avec du code ?


## Autheur:

**Florian Rascoussier** 4IF - INSA Lyon

## Tuteur:

**Lionel Brunie**, Professeur et Directeur du Département d'Informatique de l'INSA de Lyon (IF), chercheur au laboratoire LIRIS


## Jury:

**Lionel Brunie**

# Abstract

Ce rapport de projet présente un ensemble de recherches et de réflexions personnelles pour tenter d'explorer la pratique de la programmation et le code qui en résulte sous l'angle de la création artistique. L'objectif est de comprendre ce qu'est le code et comment il peut être concu et créé de manière artistique. Il s'agit donc de saisir en quoi le code est-il de l'art.

En outre, le code étant un outil à même de former des images, des mouvements ou des sons, on ne s'intéressera pas à la manière de créer des oeuvres d'art en utilisant du code. Il s'agit de se focaliser sur le code et la programmation elle-même, et non pas sur le produit ou résultat de ces derniers.

# Remerciements

Remerciements aux encadrants de ce projet, à savoir:

- **Lionel Brunie** - <lionel.brunie@insa-lyon.fr>: Encadrant final du PPH.
- **Gonzalo Suarez Lopez** - <gonzalo.suarez-lopez@insa-lyon.fr>: Premier encadrant du PPH.

Une première ébauche de ce rapport à d'abord été rédigé dans les semaines précédentes à une séance de débat. Celle-ci a permise de réunir quelques personnes du Département Informatique de l'INSA Lyon ; afin de discuter de la problématique et prendre connaissances des recherches préalables avant de débattre sur le sujet et de proposer de nouvelles pistes de réflexion.

Remerciements particuliers aux participants de la séance de débat, qui à eu lieu le Jeudi 7 avril 2022 (10-12h):

- **Eric Guérin** - <eric.guerin@insa-lyon.fr>: Professeur, chercheur au laboratoire LIRIS.
- **Anicet Nougaret** - <anicet.nougaret@insa-lyon.fr>: Etudiant IF 4ème année.
- **Ithan Velarde Requena** - <ithan.velarde-requena@insa-lyon.fr>: Etudiant IF 3ème année.

# Avant propos

Ce projet de PPH est titré par la question «Le code est-il de l'art ?». Ce titre décrit bien l'objectif et le cadre général de ce PPH mais couvre un vaste champs de réflexion qui n'est pas réellement représentatif du point de vue choisie. En effet, le code étant le produit d'une pratique, il semble assez direct de dire qui puisse être l'object d'une création ou d'une pratique artistique. Ainsi, la problématique réelle est donc «Quelles sont les possibilités de faire de l'art avec du code ?». Cette question est donc ciblée sur les pratiques, les formes et les moyens de création artistique ayant le code pour object.

La réponse à la problématique ne pouvant passer que par la compréhension du sujet global, il va donc falloir consacrer les 2 premières parties du rapport à discuter de ce qu'est le code, et de la nature de l'art, avant de réellement pouvoir aborder les différents axes de création artistiques, et les pratiques connexes que l'on peut relier au code et à la programmation. En effet, art comme code sont des concepts flous qui cachent une multitude des réalités et concepts différents. Bien qu'il ne sera pas possible d'en donner une définition précise et complète, discuter de ces concepts est néanmoins nécessaire pour se doter des outils de compréhension et de réflexion utiles pour la suite du rapport.

Enfin, il semble important de rappeler que ce rapport est issue d'un projet de PPH, c'est à dire d'un travail personnel de $4^{\text{ème}}$ année, sans créneau horaire réservé dans la formation, et valorisé au minimum d'un unique crédit ECTS. Il est donc important de considérer que ce rapport n'a en aucune manière l'ambition de faire un tour exhaustif de la question auquel il tente d'apporter un éclairage.

# Contents

# 1 Introduction

Nous verrons d'abord comment la pratique de la programmation et le code qui en résulte ont évolué afin de comprendre en détail qu'est ce que le code. Nous nous intéresserons ensuite à l'art, et la relation entre le code et ce dernier. Enfin, nous exploreront les diverses méthodes, pratiques et techniques qui font du code un art à part entière.

# 2 Qu'est-ce que le code ?

## 2.1 Programmation, code et langage

La programmation est un terme générique. Composé du suffixe *-ation*, du latin *-atio*, utilisé pour signifier une action, et du nom *programme* lui-même issue via le latin *programma* du grec ancien πρόγραμμα. qui peut désigner divers concepts selon le domaine, par exemple le terme *programmation* désigne, dans le milieu du cinéma, l'action de déterminer les programmes ou films d'une salle donnée. Bien sûr, le terme est ici considéré sous l'angle des sciences informatiques. «Pour éviter cette confusion polysémique, des termes alternatifs sont parfois utilisés, comme le code ou le codage, mais la programmation informatique ne peut pas se réduire au code» [1]. D'après le Larousse, la programmation est donc: «

- l'action de programmer, c'est-à-dire fournir à un ordinateur les données et les instructions conscernant un problème à résoudre, une tâche à effectuer, etc.
- l'établissement d'un programme, c'est-à-dire une séquence d'instructions et de données enregistrée sur un support et susceptible d'être traitée par un ordinateur.

» [2]. Cette définition n'est pas parfaite mais pose les éléments essentiels. Elle montre que la programmation est une action, un acte qui suppose des considérations techniques (quelle machine, quelle format, quels instructions), physique ou structurelles (comment l'exécution est effectuée, comment fournir les instructions à la machine, quel support), et un but (quelle tâche, pour quoi faire). La programmation est donc une activité créatrice et créative. On comprends aussi que le code est l'object de cette création, ce qu'elle produit.

Le code est donc un object issue d'un acte de production. En cela, la programmation relève donc de la technique tandis que le code est de l'ordre de la production. Le mot *code* est cependant lui aussi fortement polysémique. En effet le terme est issue du latin *codex*, variante de *caudex* signifiant «tronc». Le terme est cependant fortement associé aux livres et particulièrement aux ouvrages juridiques. Aujourd'hui, le terme peut prendre divers sens selon le domaine considéré, du droit à la linguistique en passant par l'informatique ou la cryptographie. Le Larousse distingue ainsi de nombreux sens différents pour le mot, dont les plus intéressantes du point de vue du sujet sont les suivantes: «

- Ensemble de règles qu'il convient de respecter.
- Système de symbole permettant d'interpréter, de transmettre, un message, de représenter une information, des donnée.
- Système conventionnel, rigoureusement structuré, de symboles ou de signes et de règles combinatoires intégrées dans le processus de la communication.
- Ensemble d'instructions écrites dans un langage lisible par l'homme et devant être traduites en langage machine pour être exécuté par un ordinateur.

» [2]. Toutes ces définitions sont intéressantes, notamment en considérant le point de vue d'un non informaticien. On comprend que le code désigne donc un moyen de transmettre de l'information via un système de règles, mais aussi les règles elle-mêmes.

On considèrera tout de même quelques définitions du Wikitionnaire: «

- Le code source est un texte qui représente les instructions de programme telles qu'elles ont été écrites par un programmeur.
- Le langage machine, ou code machine, est la suite de bits qui est interprétée par le processeur d'un ordinateur exécutant un programme informatique.
- Système de symboles permettant de représenter une information dans un domaine technique (code binaire, alphanumérique, morse).

» [3]. Ces définitions viennent compléter la définition du code. Du point de vue de l'informatique, le code est donc un ensemble d'instructions et les instructions elles-mêmes qui sont alors dépendantes d'un langage, système et niveau d'abstraction

considéré. En effet, tout code ne peut être écrit qu'en respectant des normes et des règles afin d'être finalement exécutable par une machine. Il est le fruit d'un acte de création, d'un travail d'écriture et de conception. Le code est donc fortement lié aux règles qui le définissent.

Cet étallage de définitions permet donc de saisir la nature profondément polysémique du terme et son lien étroit avec la notion de programmation. On peut donc catégoriser les différents sens qui seront utiles dans la suite de ce rapport:

- Séquence instructions et de données qui constituent un programme et qui ont vocation à être exécuté par un ordinateur. Il s'agit d'un sens général qui englobe une variétés de représentation, du *code source* aux Application-specific integrated circuit (ASIC).
- Système de règles, de symboles et de conventions permettant de représenter une information. Il s'agit d'une considération générale de ce que peut désigner le terme. On pensera notamment aux langages informatiques et langages de programmation qui sont des éléments essentiel à la pratique de la programmation moderne et qui déterminent fortement la forme que prendra le code source.
- On désigne par *code source*, l'ensemble des documents qui forment un programme avant toute forme de traitement. Ce code est alors réparti dans un ou plusieurs textes et documents écrits qui peuvent être rédigés et lus par un humain. Ils servent de base à divers procédés permettant de passer de ces documents à un ensemble de données et d'instructions exécutés par un ordinateur.
- Le *code machine* désigne les instructions, généralement après traitement, qui sont donc directement exécutées par le processeur d'un ordinateur. Elle n'ont pas vocations à être écrites ou lues par un humain, et sont dites *de bas niveau*. Elles sont également fortement dépendantes du jeu d'instruction, c'est-à-dire des instructions machines éxécutables par le processeur considéré. Cette élément explique entre autre le besoin de disposer de systèmes de représentations de plus hauts niveaux, indépendants de la notion de jeu d'instruction. On notera que l'*assembly*, c'est à dire du code machine écrit sous une forme lisible par un humain n'en est pas stricto sensu une forme alternative puisqu'il représente la même chose, mais nécessite quand même une conversion.
- Enfin, on introduira la notion de *code-idée*, c'est à dire l'algoritme, l'objectif derrière le code source ou machine, l'idée indépendante de tout langage.

Cette idée traverse les différents niveaux de code et nait en premier lieu dans l'esprit du programmeur lorsqu'il écrit ou lit un code source. Cette idée peut s'exprimer en une multitude de formes selon le choix du langage utilisé. D'une certaine manière, le choix du langage va cependant impacter la forme prise par l'idée et il y a donc une distinction entre l'idée, la forme et l'exécuion effective.

Pour résumer, on peut considérer que le code est un moyen de transmettre des informations, mais aussi de les représenter. Ces informations constituent un programme, fruit du travail du programmeur et conçu avec un objectif praticulier. Pour cela, il est nécessaire de respecter des règles généralement sous la forme d'un langage de programmation. Afin de mieux comprendre l'évolution de ces concepts et leurs différentes formes, il convient de s'intéresser à l'histoire.

## 2.2 Petite Histoire du code

Remonter dans l'histoire du code et de la programmation est importante pour comprendre les différentes ses différentes formes, et ses évolutions. La programmation est une technique qui permet définir des règles et des comportements sans avoir à changer le système physique qui l'utilise. Selon la définition que l'on se donne, et les exemples que l'on choisi de considérer, on peut remonter jusque dans l'antiquité avec les automates d'Héron d'Alexandrie [4]. L'idée originale étant de pouvoir définir un système capable de modifier son comportement sans modifier ses composants physiques dans le but évident de multiplier les comportements sans avoir à changer le système.

La première invention majeure remonte cependant au début du XIX$^{\text{ème}}$ siècle, avec l'invention du métier à tisser Jacquard, en 1801 à Lyon [5]. Son invention, inspirée des orgues de barbarie, permettait de lever automatiquement les fils de soie nécessaire à la réalisation de motifs. Motifs pouvant être modifiés grâce à un système modulaire de cartes perforées et d'un mécanisme en carré mobile [6]. Ce système est donc l'ancêtre direct des cartes perforées encore en usage au XX$^{\text{ème}}$siècle, et successivement amélioré par les équipes de Sir Thomas Watson et sa société IBM. Passant par étape de 22 à 80 colonnes et de 8 à 10 lignes, la fameuse *IBM card* a permis à l'entreprise de se développer fortement dans la première moitié du XX$^{\text{ème}}$ siècle, en restant pendant près de 40 ans le moyen par excellence pour stocker, transmettre et traiter des données [7].

Entre temps, il est important de regarder les developpement de Charles Baddage

et ses travaux sur les machines calculatoires et analytiques qui font qu'il est souvent considéré, non sans raison, comme le père de l'ordinateur [8]. Inspiré par le métier Jaquard, ce professeur de l'université de Cambridge a d'abord créé le *Difference Engine*, une machine entièrement analogique, composée de pièces mécaniques et capable de calculer automatiquement des tables de calculs pour certaines fonctions mathématiques comme le logarithme, ou encore pour le calcul automatique des marées. Bien qu'il n'acheva jamais sa machine, il proposa une nouvelle idée d'une machine généraliste qui aurait eu sa propre unité centrale de calcul et sa mémoire.

Dans les notes accompagnants la traduction de notes d'un séminaire donné par Baddage en 1840, Augusta Ada King, comptesse de Lovelace, ou simplement Ada Lovelace proposa une méthode de calcul automatique de la suite des nombres de Bernouilli sur la machine analytique, sous forme d'un diagramme et de notes. Cette méthode décrite dans la note G, qui se distingue du pur calcul scientifique jusque là envisagé, notamment par Baddage, et est souvent considéré comme le premier véritable programme informatique [9]. Ada Lovelace est ainsi considéré comme la première programmeuse de l'histoire.

Il faut cependant attendre le milieu du XXème siècle, et notamment les travaux du mathématicien anglais Alan Turing et son article fondateur de la science informatique, pour que cette technique se développe réellement "On computable numbers, with an application to the entscheidungsproblem". La programmation allant de paire avec le développement et la monté en puissance des ordinateurs, on passe progressivement des programmes simples réalisés physiquement sur circuit électroniques, aux programmes sur cartes perforées, avant que ne se développent les premiers langages de programmations.

TODO: histoire des langages de programmation

## 2.3 Le Langage

Le terme *langage* est souvent associé en premier lieu à une faculté intrinsèque de l'homme. En informatique, il désigne la façon dont instructions et données sont codées et de les manipuler. Le langage a enfin un sens différent du point de vue de l'artiste. Aborder le sujet du langage et du code requiert ainsi de bien comprendre les notions couvertes par le terme en question.

Pour le dictionnaire Larousse, le langage à donc de multiples définitions: «

- Faculté propre à l'homme d'exprimer et de communiquer sa pen-

sée au moyen d'un système de signe vocaux ou graphiques; et ce système.

- Système structuré de signes non verbaux remplissant une fonction de communication.
- Ensemble des procédés utilisés par un artiste dans l'expression de ses sentiments et de sa conception du monde.
- Mode d'expression propre à un sentiment, à une attitude.
- Ensemble de caractères, de symboles et de règles permettant de les assembler, utilisé pour donner des instructions à un ordinateur.
- (machine) Langague directement exécutable par l'unité centrale d'un ordinateur, dans lequel les instructions sont exprimées en code binaire.

» [2]. On remarque que ces définitions recoupent en parties celles que l'on a pu voir dans les parties précédentes au sujet du code. Code et langage sont ainsi très lié. Du point de vue du programmeur, le langage est en effet la langue qui définit comment organiser les instructions et les données afin de produire un programme [11].

Un langage de programmation est un outil pour le programmeur. En effet, tout programme étant par définiton une suite d'instructions machines c'est-à-dire de 0 et de 1, il n'est pas aisé pour un humain d'écrire ses programmes directement dans ce langage bas niveau. Cependant, il est possible d'écrire des programmes dans des langages compréhensible par un humain mais qui puisse être tout de même être transformé en langage machine pour être exécuté par un ordinateur. De mêmes que les langues parlées, il existe une multitude de langages de programmation. Comme le dit Amade, «Il existe des pratiques de programmation très différentes, il existe aussi des langages de programmation qui proposent des modes d'approche très différentes» [11]. La variété des premières expliquant naturellement la multiplicité des seconds, en plus d'autres facteurs comme les évolutions de la recherches, l'histoire jusque même aux goûts, habitudes et usages des programmeurs.

Le choix du ou des langages que l'on souhaite utiliser pour écrire un programme est donc un aspect important. Comme l'écrit Dijkstra dans son livre *A Discipline of Programming* en 1976: «[...] one is immediately faced with the question: Which programming language am I going to use ?, and this is not a mere question of presentation! A most important, but also most elusive, aspect of any tool is its influence on the habits of those who train themselves in its use. If the tool is a programming language, this influence is, -whether we like it or not- an influence on our thinking habits.» [12].

La création d'un langage de programmation est en elle-même un tâche complexe. En effet de très nombreux éléments rentrent en comptent dans la création d'un langage. Comment les rappellent les auteurs de *Introduction à la théorie des langages de programmation*, «Nous sommes encore très loin d'avoir trouvé un langage de programmation définitif. Presque chaque jour, de nouveaux langages sont créés et de nouvelles fonctionnalités sont ajoutées aux langages anciens. [...] La première chose que l'on doit décrire, quand on définit un langage de programmation, est sa syntaxe. Faut-il écrire x := 1 ou x = 1? Faut-il mettre des parenthèses après un if ou non ? Plus généralement, quelles sont les suites de symboles qui forment un programme? On dispose pour cela d'un outil efficace : la notion de grammaire formelle. À l'aide d'une grammaire, on peut décrire la syntaxe d'un langage de manière qui ne laisse pas de place à l'interprétation et qui rende possible l'écriture d'un programme qui reconnaît les programmes syntaxiquement corrects. Mais savoir ce qu'est un programme syntaxiquement correct ne suffit pas pour savoir ce qui se passe quand on exécute un tel programme. Quand on définit un langage de programmation, on doit don également décrire sa sémantique, c'est-à-dire ce qui se passe quand on exécute un programme. Deux langages peuvent avoir la même syntaxe mais des sémantiques différentes.» [13]. Un langage de programmation est ainsi composé plusieurs éléments qui permettent à une machine de comprendre et d'excuter le programme sans ambiguité.

En plus ces considérations, les langages se définissent par la ou les grands approches qu'ils choisissent de considérer ou favoriser. Ce sont les paradigmes. Dans la conférence intitulée *L'importance des langages en informatique* en Nov. 4, 2015 à l'INRIA de Rennes, Berry reviens sur le processus créatif derrière la création des nombreux langages. Au départ, on peut considérer deux langages A et B différents avec des approches fondamentalent uniques l'un par rapport à l'autre. Chacun forme sa propre communauté d'utilisateur. Puis arrive un nouveau langage C qui ne propose pas un nouveau point de vue mais reprend les concepts des deux langages précédents pour tenter tant bien que mal de les associéer. Ce dernier viens grapiller des utilisateurs aux deux premiers. Les langages A et B originaux se mettent donc à incorporer des concepts de l'autre afin de récupérer des utilisateurs et venir proposer de nouvelles améliorations à ses utilisateurs.«C'est-à-dire que vous avez des tas de gens qui ont des idées complètement baroques. Vous avez des tas de groupes d'utilisateurs avec des traditions totalement différentes dans des pays qui n'ont rien à voir [...] et la paysage deviens très joyeusement incompréhensible. Et c'est là vie, parce que c'est un espace de création. Difficile.» [14]. Berry propose ainsi une vision caricaturale mais néanmoins descriptive derrière l'apparition et l'évolution des langages informatiques.

Le langage, support essentiel de la programmation moderne, est donc en lui-

même un «espace de création» [14] qui n'est pas sans influence sur le code en lui-même [12]. Au contraire, il représente un élément essentiel du code, de par la structure, la forme et finalement la philosophie qu'il impose nécessairement de considérer. Nous devrons donc explorer ce que cela implique sur le plan artistique. Cependant, il faut d'abord nous intéresser à la question de l'art lui-même.

# 3 Le code est-il de l'art ?

## 3.1 L'art

Bien difficile est le problème de la définition de l'art. Faut-il tenter d'en donner une définition précise, chercher à décrire ce qu'il n'est pas ou encore tenter d'apporter une certaine idée de la pensée qui l'inspire et le fait naître ? C'est que le terme peut facilement changer de sens d'une personne, d'un groupe, d'un pays ou d'une culture à l'autre.

## 3.2 La programmation en tant que pratique

## 3.3 Le code comme medium

# 4 Faire de l'art avec du code ?

## 4.1 Code et poésie

Le code se définissant le plus souvant comme étant un texte, il y a alors un parallèle naturel à faire entre texte litéraire ou poétique et texte de code informatique [15].

# 6 Appendices

## 6.1 Obscurcissement du code

# Acronyms

**ASIC** Application-specific integrated circuit. 3

# Références


[1] M Romero, B Lille, and A Patiño. *Usages créatifs du numérique pour l'apprentissage au XXIe siècle*. Presses de l'Université du Québec, 2017.

[2] Claude Nimmo and Larousse (Firm). *Le petit Larousse illustré*. 21, rue de Montparnasse 75283 Paris Cedex 06: Larousse, 2017.

[3] *code*. fr. `https://fr.wiktionary.org/wiki/code`. Consulté le 19-3-2022.

[4] View all of Hansel's posts. *The history of coding and computer programming*. en. Accessed: 2022-3-21. Aug. 2018.

[5] *Les inventions qui ont changé le monde*. 1st Ed. Edition Sélecion du Reader's Digest, 1982. ISBN: 2-7098-0101-9.

[6] *Jacquard - Les Rues de Lyon*. fr. Accessed: 2022-3-21. July 2009.

[7] *The IBM punched card*. en. Accessed: 2022-3-21. Mar. 2012.

[8] B Jack Copeland. "The modern history of computing". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N Zalta. Winter 2020. Metaphysics Research Lab, Stanford University, 2020.

[9] Joasia Krysa. "Ada Lovelace: There Never Was a Note G". In: (). Ed. by Carolyn Christov-Bakargiev. introduction to a notebook 55 Ada Lovelace, author Joasia Krysa, published in dOCUMENTA (13) series 100 Notes – 100 Thoughts. Also published as a chapter in The Book of Books.

[10] Alan Turing. "On computable numbers, with an application to the entscheidungsproblem". In: *Proc. Lond. Math. Soc. (3)* s2-42.1 (1937), pp. 230–265.

[11] Bernard Amade. *Introduction à la programmation*. Paris: MA Editions, 2019.

[12] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976, pp. vii–30.

[13] Gilles. Dowek and Jean-Jacques Levy. *Introduction à la théorie des langages de programmation*. Ellipses, 2006.

[14] *L'importance des langages en informatique*. INRIA de Rennes, Nov. 4, 2015.

[15] Florian Cramer. "Digital code and literary text". In: *Beehive Hypertext/Hypermedia Literary Journal* (Sept. 27, 2001).

# Bibliographie complémentaire


[16] Andy Oram and Grew Wilson. *L'art du beau code*. 1ère Ed. Paris: Edition O'REILLY, 2008. 621 pp. ISBN: 978-2-84177-423-4.

[17] Martin Fowler. *Refactoring*. 2nd Ed. Malakoff: Dunod, 2019. 419 pp. ISBN: 978-2-10-080116-9.

[18] Robert C. Martin. *Clean Code: a handbook of agile software craftsmanship*. Montreuil: Pearson, 2009. 457 pp. ISBN: 978-2-3260-0227-2.

[19] Dustin Boswell and Trevor Foucher. *The art of readable code*. 1st Ed. O'Reilly Media, Inc, 2011. 204 pp. ISBN: 978-0596802295.

[20] Donald E. Knuth. *The art of computer programming*. 3rd Ed. Vol. 1. Addison-Wesley Longman, 1997. 664 pp. ISBN: 0-201-89683-4.