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

Problématique

Le code est-il de l'art, et 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)

Semestre d'été 2022

Problématique

Le code est-il de l'art, et quelles sont les possibilités de faire de l'art avec du code ?

Auteur :

Florian Rascoussier, étudiant en 4ème année d'informatique à l'INSA Lyon

Tuteur :

Lionel Brunie, Professeur et directeur du département d'InFormatique de l'INSA Lyon (IF), chercheur au laboratoire LIRIS

Jury :

Lionel Brunie

Éric Guérin, Professeur à l'INSA Lyon, chercheur au laboratoire LIRIS

Abstract

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

Considérer la question implique de bien comprendre ce que l'on entend par code. On comprend que le code est un terme qui cache plusieurs éléments, notamment le *code-source*, et qu'en général on considère le code un texte composé d'instructions dans un langage donné. Il faut aussi discuter de la question de l'art. Celle-ci est complexe, mais on peut se donner un ensemble de définition prenant en compte les réactions qu'il suscite et la démarche qui l'accompagne. On peut alors tenter de comprendre comment le code peut être utilisé pour créer des œuvres d'art et comment la programmation à l'origine de ces dernières peut être considérée comme un art. La programmation étant une activité créative par excellence, il est possible d'en faire une activité artistique. De même, le code peut se rattacher à la poésie et à la littérature. Les exemples surprenants ne manquent pas. Enfin, nous verrons certaines de ces pratiques ainsi que l'impact de cet art sur la culture moderne et la société. En effet, bien qu'encore principalement cantonné à des aspects pratiques, le code a une esthétique et un univers particulier qui fascine et influence les artistes dans d'autres disciplines.

En annexe, certains sujets seront abordés différemment. Un exemple complet de programmation artistique sous le prisme de l'obfuscation est présenté. On pourra également retrouver de nombreux exemples de codes artistiques. On retrouvera ainsi divers programmes parfois célèbres, et des *codésies* (ou *codeworks*), qui sont des textes poétiques mêlés ou formés de code.

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 a 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 dans un premier temps. La seconde partie ayant permise de débattre sur le sujet et de proposer de nouvelles pistes de réflexion.

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

— **Éric Guérin** - <eric.guerin@insa-lyon.fr> : Professeur, chercheur au laboratoire LIRIS.
— **Anicet Nougaret** - <anicet.nougaret@insa-lyon.fr> : Étudiant IF 3$^{\text{ème}}$ année.
— **Ithan Velarde Requena** - <ithan.velarde-requena@insa-lyon.fr> : Étudiant IF 4$^{\text{ème}}$ année.

Remerciements aux personnes présentes lors de la soutenance du PPH qui a eu lieu le vendredi 13 mai 2022, dans l'amphithéâtre Emilie du Châtelet, à la bibliothèque Marie Curie, particulièrement à **Anicet Nougaret** dont certains transparents ont servi pour la présentation.

Encore un grand merci à mon tuteur **Lionel Brunie** pour sa confiance et sa patience tout au long de ce projet et au-delà, ainsi qu'à toutes les personnes avec qui j'ai pu échanger sur le sujet.

Je dédie ce travail à mes amis, à **Salomé Voltz**, **François Mutti**, **François Vandermersch**, **Laétitia Getti**, **Dimitri Lazarević** et bien d'autres encore pour leur soutien au cours du projet, et à la mémoire de **Thomas Garcia**.

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 champ de réflexion. Ainsi il a d'abord été envisagé de se restreindre à la question connexe : *Quelles sont les possibilités de faire de l'art avec du code ?*. Cette question est davantage ciblée sur les pratiques, les formes et les moyens de la création artistique ayant le code pour objet. Cependant, au fil des recherches et des avancées du projet, il est apparu que l'approche finalement retenue était plus large en se rapprochant plus de la question initiale. Finalement, la problématique est donc un assemblage des deux questions : *Le code est-il de l'art et quelles sont les possibilités de faire de l'art avec du code ?*. En réalité, cela revient à la première question, mais souligne les transformations du projet.

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 pouvoir aborder les différents axes de création artistiques, et les pratiques connexes que l'on peut relier au code ou 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 absolue, 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. Enfin, il semble pertinent de mentionner que ce projet a très largement dépassé les quelque 25 heures conseillées pour la réalisation d'un PPH, et qu'il a obtenu la note de 19/20 malgré un rapport non finalisé au moment de l'évaluation.

Table des Matières

Codes et programmes

1 Introduction

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

2 Qu'est-ce que le code ?

Comprendre le code est la première étape indispensable afin de pouvoir considérer le problème du rapport entre celui-ci et l'art. Le terme n'est pas nécessairement bien connoté dans la sphère de l'informatique de même que l'on ne parle pas de rond, mais de cercle en géométrie cartésienne. Pourtant, le terme est intéressant en cela qu'il a le mérite d'être en apparence simple, à même de parler à chacun. Il faut cependant tenter d'éclaircir le terme et tenter d'en apporter une définition satisfaisante.

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» [3]. D'après le Larousse, la programmation est donc : «

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

» [2]. Ces quelques définitions ne sont pas parfaites, mais posent certains éléments essentiels. Elles montrent que la programmation est une action, un acte qui suppose des considérations techniques (quelle machine, quel format, quelles 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 comprend aussi que le code est l'objet de cette création, ce qu'elle produit.

Le code est donc un objet issu d'un acte de production. En cela, la programmation relève donc de la technique tandis que le code est de l'ordre du produit, du résultat. Le mot *code* est cependant lui aussi fortement polysémique. En effet, le terme est issu du latin *codex*, variante de *caudex* signifiant «tronc». Le terme est historiquement associé aux livres et particulièrement aux ouvrages juridiques. Aujourd'hui, il 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 données.
- 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 compréhensible et lisible par l'homme, 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 elles-mêmes.

On considèrera tout de même quelques définitions du Wiktionnaire : «

- Le code source est un texte qui représente les instructions d'un 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, etc).

» [4]. Ces définitions viennent compléter celle 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 étalage 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 variété de représentation, du *code source* aux ASICs.
— 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 essentiels 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. Elles 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 exécutables par le processeur considéré. Cet 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'algorithme, l'objectif derrière le code source ou machine, l'idée indépendante de tout langage ou implémentation. 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'exécution 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 particulier. 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 leur histoire.

2.2 Petite Histoire du code

Remonter dans l'histoire du code et de la programmation est important pour comprendre ses origines, ses différentes formes et ses évolutions. La programmation est une technique qui permet de 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 de Héron d'Alexandrie [20]. 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 effets sans avoir à changer le système qui les produit.

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 [17]. 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 [18]. 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 [19].

Entre temps, il est important de regarder les développements du mathématicien 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 [21].

Inspiré par le métier Jacquard, ce professeur de l'université de Cambridge a d'abord créé le *Difference Engine* avant d'envisager l'*Analytical Engine*. Cette machine entièrement analogique est théoriquement 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 accompagnant la traduction de notes d'un séminaire donné par Baddage en 1840, Augusta Ada King, comtesse de Lovelace, ou simplement Ada Lovelace proposa une méthode de calcul automatique de la suite des nombres de Bernoulli sur la machine analytique, le tout sous forme d'un diagramme et d'informations complémentaires. 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 [22]. Ada Lovelace est ainsi considéré comme la première programmeuse de l'histoire.

Il faut cependant attendre le milieu du XX$^{\text{ème}}$ siècle, et notamment les travaux du mathématicien anglais Alan Turing et son article fondateur de la science informatique : « On computable numbers, with an application to the entscheidungsproblem », pour que cette technique se développe réellement [1]. La programmation allant de pair avec le développement et la montée en puissance des ordinateurs, on passe progressivement de programmes simples réalisés physiquement sur circuit électroniques, aux programmes sur cartes perforées, avant que ne se développent les premiers langages de programmations. Ces langages ont largement impacté la pratique de la programmation du fait du cadre et des outils qu'ils donnent aux programmeurs.

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 les règles permettant 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 pensée au moyen d'un système de signe vocal ou graphique ; 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) Langage 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 [6].

Un langage de programmation est un outil pour le programmeur. En effet, tout programme étant par définition une suite d'instructions machines, soit 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 humainement lisibles, mais qui puisse être tout de même être transformé en langage machine pour être exécuté par un ordinateur. De mêmes que pour 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 différents» [6]. La variété des premières explique naturellement la multiplicité des seconds, en plus d'autres facteurs comme les évolutions de la recherche, l'histoire et jusqu'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.» [5]. Le langage n'est donc pas qu'un simple outil. Son utilisation a non seulement un impact sur la création finale, mais influence aussi fortement le créateur. Iverson cite ainsi le mathématicien George Boole : «That language is an instrument of human reason, and not merely a medium for the expression of thought, is a truth generally admitted.» [49].

Dans son article « Notation as a Tool of Thought », IVERSON décrit les principes et les considérations à garder en mémoire pour comprendre ce qu'est un langage de programmation. Il montre que c'est un instrument au raisonnement humain, en plus d'un moyen d'expression de la pensée. Il rappelle l'importance des notations en mathématiques et montre que les langages informatiques font preuve d'une plus grande universalité grâce à l'exécutabilité de leurs instructions. «... mathematical notation has serious deficiencies. In particular, it lacks universality, and must be interpreted differently according to the topic, according to the author, and even according to the immediate context. Programming languages, because they were designed for the purpose of directing computers, offer important advantages as tools of thought. Not only are they universal (general-purpose), but they are also executable and unambiguous. Executability makes it possible to use computers to perform extensive experiments on ideas expressed in a programming language, and the lack of ambiguity makes possible precise thought experiments.». Il rappelle quand même que par sa rigidité, un langage de programmation offre évidemment moins de souplesse et de possibilités que les mathématiques. Ce problème n'en n'est pas vraiment un puisqu'on pourra toujours utiliser les mathématiques pour décrire un programme, et qu'en multipliant les langages de programmation, ont leur permet ainsi de se spécialiser dans la réalisation de certaines tâches.

La création d'un langage de programmation est donc en elle-même une tâche complexe. En effet, de très nombreux éléments doivent être pris en compte. Comment le rappelle 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.» [8]. Un langage de programmation est ainsi composé plusieurs éléments qui permettent à une machine de comprendre et d'exécuter le programme sans ambiguïté.

En outre, les langages se définissent par la ou les grandes approches qu'ils choisissent de considérer ou favoriser. Ce sont les paradigmes. Dans la conférence intitulée *L'importance des langages en informatique* en 4 nov. 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 fondamentalement 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 associer. Ce dernier vient grappiller 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 le paysage deviens très joyeusement incompréhensible. Et c'est la vie, parce que c'est un espace de création. Difficile.» [7]. Berry propose ainsi une vision caricaturale, mais néanmoins descriptive derrière l'apparition, l'évolution et le foisonnement des langages informatiques.

Qui dit langage dit également dialogue. Et en effet, les langages de programmation permettent un dialogue. Dialogue entre un programmeur et une machine, mais aussi entre programmeur et lecteur postérieur de ce code, qui pourrait très bien être le programmeur initial lui-même dans le futur. Le langage, en tant que support essentiel de la programmation moderne, est donc en lui-même un «espace de création» [7] qui n'est pas sans influence sur le code en lui-même [5]. 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 ?

Une fois que la difficile question de la définition du code a été discutée, il reste encore la non-moins difficile question de la définition de l'art. La partie suivante tenter de monter toute la complexité de l'entreprise en essayant néanmoins d'établir un cadre utile pour les réflexions futures.

3.1 Un problème de définition

Bien difficile est le problème de la définition de l'art. Faut-il tenter d'en donner une version 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. Ainsi, il n'y a pas de définition précise de l'art [15]. Plutôt que chercher à en donner une définition, il vaut donc mieux se concentrer à sa compréhension et à l'étude de l'évolution de ses transformations.

Pour DAVIES dans *Definitions of Art*, l'art est une notion à la perception évolutive en fonction des époques et des auteurs : «In the past, art has been variously defined as imitation or representation(Plato 1955), as a medium for the transmission of feelings (Tolstoy 1995), as intuitive expression (Croce 1920) and as significant form (Bell 1914). Judged as essential definitions, these are unsatisfactory.» [15].

La définition de l'art et la distinction entre art et non-art a historiquement été contrôlée par les instances de pouvoir, politiques et religieuses, mais aussi par les innovations des artistes eux-mêmes. Par exemple, l'artiste Marcel Duchamp, bien connu pour ses *ready-mades* et notamment sa *fontaine* qui n'est qu'un simple urinoir, se fait connaître en montrant que c'est l'idée et la démarche qui vont compter. En d'autres termes, ce qui va définir qu'un objet est artistique, n'est pas l'objet lui-même, mais l'idée que celui-ci représente une forme d'art. Dès lors que l'artiste le décide, et que le spectateur le regarde comme tel, alors l'objet sera considéré comme artistique comme l'explique Arthur Danto dans *La transfiguration du banal : une philosophie de l'art* [23].

Malgré de ses évolutions, l'art reste un moyen d'expression, un langage à part entière avec des élaborations formelles et une force expressive signifiante. Il sert donc à communiquer comme illustré par le travail et les recherches de nombreux artistes. Par exemple, Christopher Pillault, artiste contemporain atteint d'autisme, se sert de l'art pour communiquer plutôt que de chercher à s'exprimer oralement. L'art peut permettre de transmettre des informations claires et ordonnées ou à l'inverse rechercher le mystère et l'incompréhensible. L'art est un langage métaphorique, composé de connotations et de symboles, et qui s'appuie sur des analogies.

La question de la définition de l'art pose en retour de nombreuses interrogations connexes. Parmi celles-ci se trouve celle de l'œuvre. Une œuvre se défini comme étant une chose matérielle ou immatérielle fabriquée par l'homme mais n'ayant aucune utilité fonctionnelle d'après le philosophe contemporain Heidegger, expert

de la pensée rationnelle et de la mécanique philosophique. On peut citer l'exemple du tableau de René Magritte *La trahison des images (Ceci n'est pas une pipe)* qui cherche à représenter l'objet mais pas à l'utiliser, comme l'énonce la maxime «Ceci n'est pas une pipe» du tableau [14]. Le tableau en lui-même étonne, en jouant sur les règles implicites de la pensée et du rapport entretenu entre les choses, concepts et représentations. Pour le Larousse, une œuvre au sens relatif à l'art est une «production de l'esprit, du talent ; écrit, tableau, morceau de musique, etc., ou ensemble des productions d'un écrivain, d'un artiste : Les œuvres de Bach. Une œuvre d'art. Une thèse sur l'œuvre de Rimbaud.» [26]. Cette définition est plus générale et ne se rattache pas directement à la notion d'art. Il s'agit plutôt d'une production, d'une création issue d'une pensée d'un auteur, d'un créateur ou d'un artiste.

On le comprend, l'histoire de l'art est pleine de chamboulements et de transformations de la compréhension de ce qu'il est censé être. On peut alors penser que l'art ne peut alors être défini que par sa constance inhérente au changement et à la transformation. C'est le cas de Marcus Steinweg dans « Nine Theses on Art » qui explique : «art asserts a consistency owed to its opening to inconsistency» (l'art affirme une consistance due à son ouverture à l'inconséquence). En abordant dans son essai que toute définition considérée, chacune est arbitraire puisqu'elle s'attache à ne prendre en compte qu'un ensemble restreint d'inconsistances dans une certaine perception de l'art, il montre que toute définition qui n'aurait pas pour sujet cette inconsistance même est nécessairement lacunaire donc arbitraire. Cette définition de l'art semble alors n'être pas réfutable en se rapprochant à sa manière du *conatus* de René Descartes, de l'idée que toute chose est vouée au changement. Mais cette définition ne semble pas réellement satisfaisante en dépit d'être difficilement réfutable.

Définir une relation entre art et œuvre est donc importante, mais là encore difficile. L'art étant par essence un langage mystérieux, influencé à la fois par la pensée de l'artiste et celle du spectateur, celui-ci est donc source de questionnement. Il invite au dialogue intérieur, politique ou social de par la variété des interrogations, des ressentis et des réactions qu'il suscite. Du fait qu'il convoque les sentiments, l'art et la perception de l'art est avant tout personnel. Sans tomber dans l'écueil du relativisme, il faut pourtant reconnaître que la définition de l'art est donc flexible et surtout dépendante de l'individu dont la subjectivité offre en elle-même une certaine légitimité dans la compréhension, l'appréciation et la création de l'art.

3.2 Définir Art et Œuvre

Définir l'art, de même que définir la relation entre art et œuvre est difficile. Dans la partie précédente, bien que nous ayons tenté de présenter l'étendue des possibles et la complexité des réponses, n'a pas permise de dégager un consensus. Afin d'éviter de tomber dans le relativisme, il va donc falloir se donner des règles et des définitions un minimum satisfaisantes.

En reconnaissant l'impossibilité de définir consensuellement les concepts d'art et d'œuvre, il semble cependant correct de choisir un sous-ensemble de définitions qui puissent convenir au traitement du sujet initial. Ce choix est nécessairement arbitraire et biaisé, mais est utile par la suite du rapport. En outre, ils se basent en partie sur des définitions issues de dictionnaires tels que le Larousse et le Wiktionnaire.

C'est ainsi que l'on définira pour la suite le concept d'*art* de plusieurs manières. On peut d'abord s'inspirer des définitions courantes :

- «Méthode pour faire un ouvrage, pour exécuter ou opérer quelque chose selon certaines règles.» [25]
- «Reproduction par la main de l'homme ou la représentation de ce qui est dans la nature ; par opposition à naturel.» [25]
- «Ensemble des procédés, des connaissances et des règles intéressant l'exercice d'une activité ou d'une action quelconque.» [27]
- «Manière de faire qui manifeste du goût, un sens esthétique poussé» [27]
- «Création d'objets ou de mises en scène spécifiques destinées à produire chez l'homme un état particulier de sensibilité, plus ou moins lié au plaisir esthétique» [27]

Ces définitions sont intéressantes de par leur simplicité et leur rapport avec la réalité concrète ou au sens commun.

En plus de ces définitions de surface, on utilisera nos propres définitions. Avant cela, on se donnera l'expression d'*auteur-créateur* ; pour désigner toute personne humaine physique (par opposition à personne morale) qui utilise l'ensemble de ses connaissances et facultés dans un processus de création d'une chose. Ainsi, l'*art* est donc :

- État qui naît de la volonté d'un auteur-créateur du fait des caractéristiques qu'il donne à l'objet de sa création. Il naît d'une maîtrise technique, d'une volonté particulière et d'un parti pris.
- Méthodes, cadre utilisés par un auteur-créateur pour créer un objet ou une mise en scène.
- Pratique qui nécessite de faire des choix qui aboutissent à l'établissement

une création immatérielle ou réelle, éphémère ou pérenne dont la finalité peut être utile ou non mais qui peut être perçue par un spectateur.
— Manifestations, état qui naît chez un spectateur d'une œuvre d'art et qui se caractérise par un ensemble de phénomènes (interrogations, préférences, débat, introspection, contemplation, surprise, etc) chez ce dernier.
— Élément considéré comme ayant des caractéristiques particulières qui font naître des sentiments chez un spectateur. Il suppose un dialogue entre œuvre et spectateur et des réactions chez ce dernier.

La notion d'art suppose en effet un dialogue. Dialogue d'abord entre un artiste et lui-même. Puis entre ce dernier et un public ou un spectateur imaginé. Et enfin entre l'œuvre elle-même et le spectateur.

Ainsi, dans notre cadre, une *œuvre* est donc simplement :
— Une création considérée comme artistique, c'est-à-dire qui fait naître chez un spectateur des réactions associées à l'art (interrogations, préférences, débat, introspection, contemplation, surprise, etc).
— Un objet qui peut être apprécié pour ses caractéristiques et qualités du fait d'un processus de création particulier.

On remarquera que les sens d'œuvre et d'art se recouvrent en partie puisque dans ce cadre, une œuvre peut être elle-même de l'art.

Il est important de rappeler qu'aucune de ces définitions n'est satisfaisante par elle-même. En outre, la somme de celles-ci ne peut pas l'être pleinement non plus. Elle est cependant utile dans le cadre précis de la compréhension du rapport entre programmation et code.

3.3 Programmation et code

Une fois que l'on s'est fixé un cadre suffisant pour comprendre les notions de programmation, code, art et œuvre, on peut alors dire simplement que **le code est à la programmation ce que l'œuvre est à l'art**. Cette phrase simple condense les parties précédentes en une maxime qui ne doit pas faire oublier les réserves et les problèmes mis au jour plus tôt.

On peut alors dire que **le code peut être de l'art dans la modalité que le code peut être œuvre et qu'œuvre peut être art**. Cette deuxième maxime est à comprendre dans les réflexions de la sous-partie précédente et notamment sur la relation sémantique entre les termes *art* et *œuvre*.

On peut comprendre ces deux maximes de différentes manières en fonction des

définitions considérées : Le code en tant que système de règles ou en tant que langage est donc de l'ordre de la méthode. En ce sens, dire que *le code est de l'art*, c'est dire qu'un ensemble de contraintes et de règles s'inscrivant dans une méthode permettent de définir un art puisque art peut être cadre et méthode 3.2. En outre, on pourrait tenter d'établir un parallèle de la question sur le rapport entre code et art, avec celui entre langage de programmation et art. Un système tel qu'un langage de programmation, du fait des choix de conceptions et d'écriture qu'il impose est en lui-même un cadre à la création artistique. Il est ainsi possible de considérer que la création de langages de programmation est aussi un art. Cependant, le langage étant ici un outil dans le cadre de la programmation, il faudrait alors aborder la question du rapport entre art et artisanat.

La programmation est donc une pratique créatrice qui impose des choix et nécessite une prise de position et un point de vue particulier. Il est naturel de la considérer comme une pratique artistique et donc comme de l'art. De même, le code en tant qu'objet, autrement dit un ensemble d'informations par exemple sous forme de texte, peut être une œuvre puisqu'il est issue d'une création artistique.

On remarque alors que la critique relative à l'utilité des œuvres, qui est traditionnellement utilisée pour discréditer le code, est en fait caduque. En effet, dire qu'une chose ne peut être de l'art au seul motif qu'elle peut avoir une finalité pratique n'a pas lieu d'être. D'une part, il existe du code conçu sans aucune intention d'usage pratique ou de finalité utile réelle. D'autre part, le rapport entre art et utilité n'est qu'une conception de l'art dont les contre-exemples sont légion et qui n'entre même plus en considération dans les définitions modernes de l'art. On peut dire que si tout code naît d'une idée et suppose un but, ce but n'est pas nécessairement celui d'une finalité utile. En outre, l'art est souvent aux cœurs d'intérêts économiques importants : du mécénat aux expositions en passant par de nouvelles variations comme les NFT.

Au regard de ces réflexions, il apparait que le code peut être de l'art. La notion de code doit alors s'entendre dans la complexité de ce qu'elle implique de même que la notion d'art. Un parallèle intéressant consiste alors à comparer le code avec une forme d'art plus ancienne et établie telle que la musique [42].

Elle désigne l'art d'arranger un ensemble de sons entre eux dans le temps [44]. Par extension, elle désigne aussi cet arrangement de son. Elle est généralement associée à une certaine idée d'appréciation et de plaisir par l'écoute. C'est une pratique encadrée par un ensemble de règles physiques et conventions [43]. La musique est donc un art de l'information. Informations qui peuvent être accédées directement par un humain en écoutant les sons lorsqu'ils sont produits par un chanteur,

un instrument ou tout autre dispositif pouvant émettre des sons. Ces informations peuvent être stockées sur un support physique, par exemple les sillons d'un disque vinyle ou les micro-polarités d'un disque dur par exemple. Elles peuvent aussi être retranscrites sous forme de texte. Texte qui peut lui-même être stocké sous différentes formes ou être lu directement par un humain disposant des connaissances et capacités adéquates. La musique se conçoit donc sur différents supports [43].

Ainsi, la musique peut désigner à la fois l'art et l'œuvre sous différentes formes de représentations. On pourrait se simplifier la tâche en dissociant les concepts derrières 2 mots : la *musique-pratique* et la *musique-œuvre*. On peut alors les comparer à la programmation et au code. En effet, de même que la musique-pratique, la programmation dispose nécessairement de règles et contraintes. Puisqu'il s'agit de programmer des machines, la programmation se voit contrainte par les principes de la physique (électronique, quantique, mécanique, etc). L'immense majorité des ordinateurs étant basés sur des calculs physiques en binaire, la programmation se doit donc de considérer d'emblée les contraintes physiques des machines en question. C'est ainsi que le programmeur doit souvent faire face à ces contraintes, par exemple en termes de vitesse d'écriture de disque, de temps de calcul, ou encore de l'espace mémoire disponible à différent niveau (cache, RAM, disque ou cloud...). La musique, elle, doit faire avant tout avec les perceptions humaines, l'acoustique et la physique des ondes sonores.

Afin d'encadrer la pratique de la musique, de nombreux cadres et règles ont étés développés. En informatique et donc en programmation, on trouve aussi de nombreux concepts et paradigmes qui offrent un cadre et un ensemble d'outils d'analyse et de construction aux programmeurs. L'objectif final est exclusivement la production d'un code dans le but d'être exécuté. Code qui se compose d'instructions à plus ou moins haut niveau en fonction des niveaux d'abstractions qui séparent le travail du programmeur de celui de la machine physique finale qui exécute les instructions. La programmation est donc aussi art de l'information. C'est aussi un certain art de la traduction. Il faut passer d'une idée humaine, exprimée en langage oral complexe et qui est souvent très imprécise et contextuelle, à un langage bien plus rigide, basés sur des règles mathématique et physiques.

De même que le musicien doit s'entraîner pendant des années afin de parvenir à la maîtrise de son instrument s'il veut pouvoir transmettre au mieux ses idées et sentiments par la musique, le programmeur doit lui aussi pratiquer pendant des années et surmonter les difficultés intellectuelles et techniques afin de mener à bien ses projets. De même que le musicien doit faire preuve de maîtrise afin de retranscrire l'idée de la musique en musique elle-même sans fausse note, le programmeur doit surmonter bugs et problèmes de développement pour passer de l'idée au code.

Et de même en musique qu'un musicien particulièrement talentueux pourra très bien surpasser de nombreux autres musiciens, un programmeur chevronné est tout à fait à même de surpasser une équipe complète. Ceci ne doit pas faire oublier que bien souvent, que ce soit dans la sphère de l'Open Source ou dans le cadre professionnel ou même amical, la programmation se pratique rarement seule mais plutôt en équipe. Il en va de même pour la musique, du solo jusqu'à l'orchestre. Le musicien peut faire alors faire de la musique son métier, tout comme la programmation peut aussi bien relever du gagne-pain que du passe-temps.

Enfin, la programmation produit un code, une fois que l'idée initiale à réussie à être exprimée d'une manière ou d'une autre en code qui peut être aisément converti en code-machine directement exécutable. Le code intermédiaire (*code-source*), exprimé d'une façon lisible pour un humain, est souvent réalisé dans un langage de programmation en particulier. Il aurait vraisemblablement une forme différente dans un autre langage de programmation. De nombreux langages ayant des propriétés similaires, le choix d'un langage plutôt qu'un autre laisse une certaine marge à la subjectivité et finalement constitue un premier choix important dans la pratique de la programmation. S'ensuit de nombreux autres choix tout au long du processus de développement, encadré par des principes, des goûts personnels, des paradigmes et des pratiques considérés comme plus ou moins bonnes. Le résultat final est donc le produit d'un processus complexe et créatif qui dispose de toutes les caractéristiques d'une œuvre. Si tous les codes ne sont pas nécessairement artistiques, un beau code, un bel algorithme ou un beau programme est parfaitement capable de susciter des réactions sur un spectateur averti ou non. De par l'aspect complexe et souvent cryptique qu'il semble avoir, il fait l'objet d'une certaine crainte mêlée de fascination sur le grand public, ce dont se sont emparées de nombreuses œuvres à l'exemple des films de science-fiction tels que The Matrix ou TRON.

4 Faire de l'art avec du code ?

Cette partie s'éloigne d'un travail de compilation et de discussion purement journalistique ou philosophique. L'art touchant à la dimension personnelle et profonde de l'individu, la section suivante prend clairement la partie d'être plus subjective parce qu'elle tente de parler d'art au sens personnel. Il est de plus difficile d'étayer tous les arguments par des sources pertinentes. Cette section tentera de présenter les considérations et les moyens de l'art du code.

4.1 La programmation en tant qu'art

Maintenant qu'on a pu établir pourquoi le code peut être de l'art, on va pouvoir discuter du comment. Il ne s'agit donc pas de s'intéresser littéralement à *faire de l'art avec du code*, mais plutôt de *faire du code avec de l'art*. En effet, on a montré précédemment que le rapport entre programmation et code se rapportait à celui qui existe entre art et œuvre et c'est donc plutôt dans le sens de la seconde proposition qu'il faut l'entendre. Dans cette partie, on s'intéressera d'abord au fond, c'est-à-dire à ce que fait ou est censé faire le code.

La programmation à donc pour principal objet l'écriture et la conception de code. Le rapport entre programmation et code est lui-même sujet à débat en fonction de l'interlocuteur et de la situation. Selon que celui-ci considère plus ou moins la pratique ou la théorie, il n'aura alors pas la même approche de la pratique. En effet, si on se place plus d'un point de vue théorique, par exemple en considérant une approche formelle de l'algorithmique et de la qualité logicielle, alors les règles que l'on va tendre à considérer sont d'ordre logique. On pourra par exemple se placer dans un certain domaine théorique, par exemple le lambda-calcul. Les considérations pratiques telles que le choix de tel ou tel langage devient alors secondaire. À l'autre bout du spectre, un programmeur de systèmes embarqué se doit de connaître au mieux les caractéristiques techniques du système en question afin de pouvoir le programmer au mieux. Dans ce cas, le choix des outils tels que le langage, les processus et outils de tests et de débogages deviennent des considérations de premier plan. Le résultat primera alors face au respect strict des règles théoriques.

La nature du projet et de l'idée initiale est bien souvent déterminante dans l'approche retenue. En outre, des considérations éloignées peuvent néanmoins amener des solutions et des approches similaires puisqu'il n'existe pas réellement de meilleure façon de faire, et que celles considérées comme potentiellement bonnes peuvent se recouper. L'ensemble des choix guidant le développement seront responsables du résultat final, du code-source produit. Il n'existe malheureusement aucune garantie de succès et un projet informatique, qu'il soit personnel ou le fruit d'années de labeurs pour des centaines de personnes peut se solder par un échec et un abandon pur et simple de celui-ci [45] [46]. Il existe ainsi de nombreux exemples de projets informatiques abandonnés à l'issue d'un échec de développement. Par exemple, le projet *IRS Tax Systems Modernization projects* aurait été mis au rebut, après 8 ans de développement et un coût estimé à plusieurs milliards de dollars dépensés par le gouvernement des États-Unis [47]. Dès lors, lorsque le code fonctionne bien, il est possible d'éprouver des sentiments positifs, de la surprise à l'émerveillement, c'est-à-dire des sentiments souvent associés à la contemplation

d'une œuvre artistique. La programmation est donc l'art de maîtrise la complexité des possibles et des contraintes afin de produire un code qui répond à des objectifs selon un point de vue particulier.

À la manière de l'écrivain qui commence immanquablement son périple par la feuille blanche et qui peut perdre courage face à l'immensité des possibles, le programmeur moderne commence souvent par un fichier vide. Au départ, il y a souvent une idée, un concept ou un objectif. Dans un cadre plus professionnel, le travail commence généralement bien en amont. Il s'agit d'abord de formaliser les attentes des clients, les besoins, les cas d'utilisations, etc. Puisqu'il s'agit de dialoguer in-fine avec la machine, rien ne doit être laissé au hasard si bien que l'étape de conception initiale peut représenter à elle seule la majeure partie du travail. Le code n'arrivant qu'en fin de cycle une fois que tous les concepts nécessaires au projet ont été formalisés.

Toutes ces étapes viennent simplifier la programmation, ou tout du moins lui fournir les bases nécessaires pour la faciliter. C'est que le programmeur doit jongler avec de nombreuses contraintes et concepts : modélisation, langages et outils, paradigmes, élégance, simplicité, efficacité, maintenabilité entre autres. Ces critères permettent de comprendre ce qu'est le *clean-code*, c'est-à-dire le beau code. Le code beau aura pour effet de rendre la vie du programmeur et de ses successeurs plus agréable et plus facile. À l'inverse, un code sera dit sale s'il ne respecte pas ces règles de beauté, ce qui aura souvent des conséquences négatives pour l'avenir du projet. Il y a donc souvent un intérêt réel à programmer élégamment. Il pourrait alors s'agir de critères purement à but fonctionnel, mais ce n'est pas le cas. En effet, certaines de ces règles reposent sur des partis pris, des réflexions et des goûts. Si elles peuvent reposer sur des réflexions réelles, elles n'en demeurent pas moins sujettes à interprétation et à évaluation. Chacun ayant plus ou moins ses propres règles et considérations de premiers plans.

En plus de la forme, il faut aussi aborder le fond, l'algorithme, c'est-à-dire le *code-idée* qui précède le *code-source*. Ils peuvent chacun être une œuvre artistique. Si l'algorithme est particulièrement ingénieux, ou à l'inverse inutilement complexe à l'extrême par exemple dans le cadre d'une compétition informatique (voir 6.3) alors il peut faire naître un sentiment de surprise, d'étonnement ou d'émerveillement pour devenir une œuvre. Cependant, bien souvent ce code se retrouvera noyé dans le reste du code-source. Il devient alors difficile de distinguer et de découvrir l'œuvre noyée dans le reste. Car comprendre la beauté d'un code n'est pas facile et demande généralement un effort significatif de compréhension. Dans le cas d'une personne qui lirait le code source rapidement sans réellement chercher à comprendre les détails de ce qu'il se passe réellement, chercher l'œuvre dans le reste du code

s'apparente à chercher un poème au milieu d'un livre de recettes de cuisine. Dans ce cas la poésie viendrait d'une recette, originale et surprenante pour un cuisinier. Cela montre malheureusement qu'il faudra lire et comprendre chaque recette afin d'en mesurer pleinement la portée artistique ou non. On trouve cependant de nombreux exemples de codes directement accessibles et qui peuvent être appréciés sans avoir à chercher (par exemple [30]).

Le code, et notamment la volonté du programmeur et le contexte dans lequel il s'inscrit peut donc lui donner toutes les caractéristiques d'une œuvre d'art. Cela n'est cependant pas automatique. Il reste à admettre que la frontière entre art et non-art peut-être difficile à déceler. Tant que l'expérience reste personnelle cela ne pose pas de problème, mais la confrontation de ses opinions avec celles d'autres personnes peut permettre de mettre au jour ou non cette frontière. Il existe cependant des moyens plus directs de déceler ou créer du code artistique.

4.2 Approche esthétique, forme et fond

Passer du fond à la forme est un moyen simple et efficace de distinguer un code banal d'une œuvre artistique. Pour le programmeur, il y a souvent une vision qui prime sur le reste et une bonne manière envisagée pour faire les choses. Selon que le programmeur soit d'emblée dans une approche volontairement artistique, cette vision entraîne dans son sillage une démarche qui peut se relever cruciale dans la concrétisation d'un projet.

De par sa complexité, le processus de programmation impose de faire des choix. Choix qui bien que guidés par une certaine idée du bien faire, font appel à des considérations arbitraires et personnelles parfois difficilement justifiables autrement que par esthétisme. Il peut en aller ainsi pour le choix d'un langage plutôt qu'un autre, d'un framework, d'une solution, d'un IDE, d'une police, etc. On pensera aux solutions JetBrains par exemple et leur aspect poussé du design qui est clairement un argument de vente. Lorsqu'on est programmeur avec des objectifs d'efficacité en tête, il faut faire attention de ne pas trop se concentrer sur ces considérations sous peine de risque de se détourner du but premier et d'utiliser beaucoup plus de ressources que nécessaire sur des aspects non directement liés aux objectifs initiaux. En effet, il peut être tentant de tout faire soi-même, selon ses propres idéaux de beauté et de forme. C'est ainsi que Donald Knuth, afin d'écrire ses livres tel que *The art of computer programming* [16] en est venu à développer de nombreux éléments afin de tout contrôler en accord avec ses propres critères esthétiques. Knuth à donc développé TeX, sur lequel est basé LaTeX avec lequel ce

présent rapport a été rédigé. Il a également créé de quoi définir sa propre police : **Computer Modern** qui est la police utilisée dans ses livres et également celle de ce document (police par défaut de LaTeX).

Mais Knuth est loin d'être le seul exemple et on pensera à l'exemple du jeu vidéo *The Witness* dont le processus de développement est en lui-même une prouesse artistique plus que technique. Le jeu a été développé pendant 7 ans par l'équipe de Jonathan Blow, qui est à la fois le concepteur, directeur, designeur et surtout programmeur principal du projet. Celui-ci refusa d'utiliser un moteur de jeu disponible sur le marché et se lança avec son équipe dans le développement de son propre moteur de jeu 3D. En effet, il souhaitait avoir le contrôle total de son jeu. De même, quand des problématiques de gestion des versions du code apparurent, il décida de développer son propre système de contrôle de version. Les zones du jeu et les éléments de ce dernier sont entièrement sérialisés en texte répartis sur des dizaines de milliers de fichiers afin de faciliter la gestion des conflits d'édition. Après le succès du jeu, Blow s'est lancé dans le développement de son propre langage de programmation et d'un moteur associé. Il travaillerait actuellement sur 2 jeux utilisant ses propres technologies [28].

Les exemples précédents sont des illustrations de l'importance que la forme peut avoir en programmation et plus généralement dans le processus de développement d'un projet informatique. L'esthétique peut recouvrir de nombreux éléments : langage, framework, librairie, IDE, moteur de jeu, typographie, choix des couleurs, etc. En outre, la programmation moderne impliquant le plus souvent la production d'un texte, on peut donc lui appliquer les mêmes techniques d'esthétique utilisées pour produire un texte par exemple en poésie. Il y a alors un parallèle naturel à faire entre texte littéraire ou poétique et texte de code informatique [9].

On pensera entre autre au calligramme qui propose un rapport intéressant sur le rapport entre la forme et le fond. En programmation, il existe notamment un exemple célèbre de code en C dont le texte prends la forme d'un cercle et qui une fois compilé et exécuté, affiche un tore en 3D grâce à du texte Cd. 2. Les exemples similaires sont nombreux et bon nombres d'entre eux proviennent de l'IOCCC, une compétition internationale de programmation C dont l'objectif est de produire le code le plus surprenant et difficile à comprendre possible. De nombreux sites permettent de se rendre compte de l'ingéniosité et de la beauté réelles de ces codes. Parmi les nombreuses propositions, on trouvera entre autre :

— Une implémentation du moteur analytique de Charles Babbage qui calcule les factorielles, les PGCD et les facteurs d'un nombre ; par Sean Barrett [31] (4)
— Un programme qui converti l'entrée standard en code morse. Le code lui-

même est composé essentiellement de DAH DIT DAH qui épelle un message en code morse ; par Jim Hague [32] (3)
— Un traducteur anglais-cochon-latin. Le code source a lui-même la forme d'une tête de cochon ; par Don Dodson [33] (7)
— Un programme dont chaque ligne de code est un palindrome. (auteur inconnu). (5)

La liste continue (voir 7).

Ces programmes se rapprochent fortement de la poésie, voir on l'ambition d'être des poèmes à part entière (9), par la forme textuelle, le rapport entre la forme et le fond, et encore au-delà, à travers leur comportement une fois le code exécuté. On peut alors se rendre compte que l'esthétique du code et celle de la littérature peuvent finir par se mélanger pour donner naissance à un texte. Ce texte, fortement inspiré par le code, reste avant tout une œuvre textuelle au sens littéraire et poétique dont la vocation première n'est plus celle de devoir être exécutable. On peut alors parler de *codework* (voir 7.2). Ce terme, vraisemblablement un néologisme inventé par Alan Sondhein par concaténation de *code* et artwork, désigne un texte mêlé de code : «Codework is a type of creative writing which in some way references or incorporates formal computer languages (C++, Perl, etc.) within the text. The text itself is not necessarily code that will compile or run, though some have added that requirement as a form of constraint.» [48].

Avec ces exemples, on peut mieux comprendre la dimension artistique que peut prendre le code. Dès lors, on comprend que le code peut être le produit d'approches esthétiques artistiques variées. Au départ réservé à un groupe d'initiés, ces nouvelles formes d'art se sont progressivement distillées dans la société notamment par internet. Le code est ainsi passé progressivement d'un mal nécessaire à une forme artistique à part entière qui regroupe ses propres courants et approches.

4.3 Univers visuels, connotations et perceptions

On l'a dit, l'essor de l'informatique et notamment la démocratisation des ordinateurs personnels, puis le développement d'Internet ont dès leur origine baignés dans la culture des sphères d'initiés à ces nouvelles technologies. On pensera par exemple au terme de *geek* et *nerd* qui se sont répandus. D'abord connotés négativement, ces termes ont peu a peu été utilisés par différentes communautés qui se sont alors appropriés ces termes. Très tôt, ces communautés ont développé leurs propres esthétiques et leur influence n'a fait que se renforcer avec l'essor des réseaux sociaux de masses. Au milieu de tous ces changements, une constante :

le développement de la programmation et l'apparition de nouveaux outils plus simples et accessible, permettant au plus grand nombre de se former sur ces sujets [5].

C'est ainsi que le code est progressivement entré dans la culture populaire. À cause de son aspect rigide et sa complexité, le code fait peur aux yeux du grand publique. Dès lors, les personnes capables de le maîtriser apparaissent alors tour à tour comme des génies solitaires ou des criminels. La contre-culture des années 1980-1990 infuse alors l'idée du délinquant encapuchonné qui terrorise entreprises et gouvernement au nom de l'anarchisme et d'un rejet de la société civile. Le modèle du hacker est né. C'est dans cette optique qu'apparaissent des films comme *The Matrix* ou *Elysium* qui font du hacker le véritable héros, car seul initié capable de lutter contre le système. Ces films ayant connu un certain succès, cette esthétique a profondément influencé la décennie suivante, particulièrement dans le milieu du jeu vidéo. La science-fiction s'est très tôt emparée de ces problématiques avec bons nombres d'œuvres de *TRON* à *Terminator*, en passant par *Jumanji : Welcome to the Jungle* ou *The Zero Theorem* [61].

L'esthétique du code s'est profondément infusée dans la culture web et les mêmes. On retrouve plusieurs esthétiques qui se croisent ou parfois s'affrontent. Puisque le code est souvent produit par une minorité, on retrouve une première connotation autour de l'espionnage, de la surveillance de masse et des complots. On se souviendra entre autre de la légende urbaine du jeu Polybius, un jeu vidéo censé causer des effets pouvant aller jusqu'à la mort, à l'IA maléfique GLaDOS dans les jeux Portal, ou au fameux Dark Net relié aux cypherpunks, espions et criminels passant par des réseaux parallèles comme Tor. De l'autre côté du spectre, on retrouvera des esthétiques plus ancrées dans la pop-culture, aux univers colorés et mélodiques. La connotation sera alors beaucoup plus positive, centrée sur l'aspect communautaire. On peut noter la prévalence du jeu vidéo, véritable catalyseur des univers développés autour ou incorporant des éléments liés au code et à la programmation.

Enfin, il semble important de mentionner l'impacte de la communauté Open Source et FOSS. Avec des figures importantes comme Linus Torvalds, programmeur bien connu pour être le créateur et programmeur principal du noyau Linux et du système de contrôle de version Git. Cette communauté prône des valeurs d'accessibilité, de contrôle raisonné des données, de décentralisation et de liberté de l'information. Cette communauté fait une utilisation importante du code y compris à des fins de liberté d'expression. Ainsi, puisque le code est un texte, celui-ci peut être imprimé et tombe sous les lois de la liberté d'expression et de la presse. Il existe plusieurs exemples frappant d'utilisation du code à des fins de contour-

nement qui pourrait être qualifiés d'artistique et qui trace un chemin direct entre code et littérature. Ainsi, Florian Cramer, dans son essai « Digital code and literary text » propose l'exemple du code source du programme PGP. Ce programme ayant été considéré comme une arme au sens légale, son auteur, Phil Zimmerman a alors eu l'idée d'en publier le code-source dans un livre. Ainsi, le programme est alors tombé dans le domaine littéraire et protégé par le *U.S. First Amendment of free speech* : «So the book could be exported outside the United States and, by scanning and retyping, translated back into an executable program» [9].

Ces communautés ont développé au fil des ans leurs propres univers et grammaires visuelles dans lequel le code prend généralement une place non négligeable. Ces esthétiques ont en retour influencé les programmeurs eux-mêmes. C'est ainsi que les outils de développement intègrent de plus en plus des notions d'esthétisme visuel, avec par exemple le thème sombre très affectionné par les développeurs pour être considéré comme plus agréable lors d'un travail généralement nocturne. De même, il est normal qu'une personne passant ses journées entières cherche à esthétiser son environnement de travail par pur plaisir ou par réel goût. La police d'écriture peut par exemple être créée et utilisée exclusivement pour certains types de code particuliers ou par certains développeurs. Ces considérations se retrouvent historiquement dans la communauté des développeurs web du fait de leur intérêt nécessairement plus élevé pour les interfaces graphiques étant donné que c'est généralement une composante importante de leur métier.

On remarque donc l'importance que le code infuse dans la culture populaire et internet moderne. Porté par diverses communautés et univers visuels, influençant les œuvres notamment audio-visuelles ou littéraires. Programmation et code sont devenues des composantes importantes de la culture actuelle.

5 Conclusion

À la question : *Le code est-il de l'art ?*, la réponse semble être : oui, le code, en tant que produit d'une technique créative peut être de l'art, ce qui ne veut pas dire que tout code est nécessairement de l'art. Discuter du sujet impose d'avoir conscience de la complexité des notions mises en jeu. D'une part le code désigne plusieurs réalités et est à mettre en relation avec la programmation. D'autre part la définition de l'art pose de nombreux problèmes. Malgré tout, le code reste un espace de création et de dialogue à part entier. L'art peut s'insinuer à différents niveaux : de complexité, de forme ou de fond, ou même graphique et visuelle. Ainsi, il existe de nombreuses potentialités et possibilités artistiques pour le code. Qu'il s'agisse

de s'imposer une démarche particulière, de viser des objectifs surprenants ou de créer un code aux propriétés esthétiques certaines, il existe de nombreuses voies artistiques pour le code. Ce code est alors à même de toucher et de provoquer des réactions ou sentiments selon les propriétés généralement attribuées aux œuvres d'art chez l'individu qui en fait l'expérience.

Code, programmation, et informatique sont des disciplines ou des objets relativement nouveaux au vu de leur histoire encore très récente, surtout en comparaison avec d'autres disciplines artistiques anciennes. Malgré tout, ces derniers ont vu un développement fulgurant et ont eu un impact absolument majeur sur l'Humanité et la vie courante de milliards d'individus. En ouvrant un nouvel espace de création artistique, code et programmation sont devenus des sujets de discussion, de débat, et de création artistique dont l'impact est déjà mesurable dans la culture actuelle. Il reste cependant de très nombreuses possibilités de développement et d'ouverture pour que le code devienne réellement un art et une forme d'expression reconnue, notamment du point de vue institutionnel et académique. À l'heure actuelle, s'il est certain que le code peut être de l'art, il est encore difficile de le considérer intrinsèquement comme tel.

6 Notes complémentaires

Les sous-sections suivantes viennent compléter sous différents aspects les propos discutés précédemment. Puisqu'il ne s'agit plus du cœur du rapport, ce sera d'accorder au sujet des prises de positions plus personnelles, détaillées et originales.

6.1 Reconsidérer l'argumentation

La lecture du rapport précédent n'a peut-être pas suffit à satisfaire le lecteur dubitatif. Et c'est bien normal. Comme on a pu s'en rendre compte, discuter d'art, et de frontière entre art et non-art impose la considération d'une myriade d'éléments. De la philosophie de l'art à l'histoire des arts, en passant par la compréhension et la sensibilité personnelle de chacun, il n'est pas aisé de dégager des consensus sur le sujet. Quel rapport entre art et utilité ? Quid de la relation entre œuvre et art ? Qu'entend-on par code ? Pour ces différentes questions, et bien d'autre encore, le rapport précédent a tenté de proposer des idées de réponses et des approches de réflexions.

Le rapport a pris pour parti de suivre une approche argumentative classique en prenant pour thèse que le code pouvait être de l'art. En reconsidérant l'argumentation faite dans le rapport précédent, on peut se rendre compte de l'établissement des propositions suivantes comme vraies :

- Une chose peut être considérée comme artistique lorsqu'elle suscite des sentiments chez ceux qui en font l'expérience sensible ou intellectuelle ; lorsque l'on attribue à cette chose des qualités qui dépassent sa simple nature ; ou lorsqu'elle est une expression de la pensée, de la culture, d'une vision et du talent de son créateur. On pourra alors dire que cette chose est une œuvre d'art.
- Un processus créatif peut être considéré comme artistique lorsqu'il peut conduire à la création d'une œuvre d'art ; ou lorsqu'il est une expression de la pensée, de la culture, d'une vision et du talent de son créateur. On pourra alors dire que ce processus est un art.
- Il existe un code tel que ce code puisse être considéré comme artistique.
- La programmation est un processus créatif par lequel découle tout code.
- Si une chose peut-être considérée comme artistique, alors le processus créatif de la chose peut être considérée comme artistique.
- Si un processus créatif peut-être considéré comme artistique, alors il peut être de l'art.
- Si un processus créatif peut être de l'art, alors tout ce qui est produit par ce

processus peut lui-aussi être de l'art.

En utilisant le Modus Ponens [40], on peut déduire par démonstration directe que le code comme la programmation peuvent être considéré comme un art. Et qu'en particulier le code peut être œuvre d'art.

En allant un peu plus loin dans le raisonnement, on pourrait aussi dire que si quelque chose peut être de l'art, alors elle l'est toujours. Cela peut sembler surprenant au premier abord. Cependant, la comparaison avec les arts établis que sont par exemple le cinéma ou la littérature tendent à montrer que cet état de fait est nécessairement vrai. En effet, que l'on aime ou pas tel ou tel film, un film est toujours une œuvre d'art. On pourrait pourtant en douter au vu des nombreux exemples étranges, choquant ou incompréhensible du cinéma. Faut-il considérer des œuvres tels que *Cannibal Holocaust* ou *Salò ou les 120 Journées de Sodome* comme des œuvres alors même que le premier est toujours interdit dans de nombreux pays tandis que le second n'a même pas pu être achevé par son réalisateur, assassiné avant la fin de la production dans ces circonstances douteuses ? Et que dire de ces innombrables films réalisés avant tout pour l'argent, de *Sharknado* à *Gods of Egypt* en passant par *The Star Wars Holiday Special* ? Et que dire encore de tous ces films simplement médiocres que le temps a déjà effacé des mémoires ?

En ce penchant de plus près sur chaque cas, on trouve pourtant des réactions, des expériences à vivre et des choses à raconter. Et c'est peut-être précisément ici, dans le dialogue et la confrontation de l'altérité que se trouve tout ce qui fait l'art de l'œuvre et l'art lui-même. Les premiers films mentionnés sont tous considérés aujourd'hui comme cultes et on en partie été réhabilités [36] [37]. Les seconds ont donné malgré eux naissance au genre du *nanar* [34]. Enfin, les derniers sont souvent à minima des reliques de leur époque et en cela digne d'intérêt de la part de la communauté des cinéphiles. En outre, on pourra aussi mentionné le cas des films perdus célèbres tel que *London After Midnight*, film perdu de 1927 et devenu culte précisément parce que personne ne peut plus le voir et dont la dernière copie a disparu dans un feu d'entrepôt en 1965 [38]. Tout cela tends donc à montrer que la proposition suivante est donc vrai :

— Si quelque chose peut être de l'art, alors elle l'est toujours.

En utilisant la logique du second ordre, on peut donc conclure des propositions précédentes que **le code est de l'art**.

6.2 Changer de logique

L'approche précédente à donc consister à appliquer le principe bien connu de la démonstration directe [39]. Une autre approche aurait été de passer par la démonstration par l'absurde, c'est-à-dire tenter de monter que dire que le code n'est pas de l'art est absurde. Cette approche différente permet d'aborder le problème sous un autre angle en offrant une alternative à la démonstration directe.

Commençons par supposer vrai la proposition suivante :
— Tout code ne peut jamais être considéré comme une œuvre d'art.
Puisque le code est le produit d'une programmation, on déduit que la programmation ne produisant pas d'œuvre n'est alors pas un art. Cette proposition est la conséquence directe de la précédente, d'après les définitions discutées en première partie du rapport. Pour montrer que la proposition supposée vraie est en réalité fausse, il suffira de trouver un contre-exemple, c'est-à-dire un cas dans lequel il apparait qu'un code est bien une œuvre, issu d'un processus créatif artistique, autrement dit d'un art.

Il va donc falloir discuter d'exemples afin de poursuivre la démonstration. Pour cela, on va pouvoir développer les exemples proposés pour la partie établissant un lien entre le code et la poésie ou la littérature (voir 4.2). Les exemples proposés proviennent de la collection de programmes C obfusqués préférés d'Enrique Bermúdez [29], professeur à la *University of Florida*. Cette collection contient 39 programmes en C, ordonnées par un titre dans l'ordre alphabétique et suivi à chaque fois d'une note d'appréciation de Enrique Bermúdez. La page intitulée *Obfuscated C Code* est également introduite par le texte suivant : «Unless otherwise indicated, the code comes from the International Obfuscated C code contest. Most files contain an explanatory text file appended to the ANSI C code. You really need to see the source code to appreciate most of these.» [30].

Sans rentrer dans les détails, on peut déjà remarquer que Enrique Bermúdez parle de sa collection préférée : («Here are some of my favorite OBFUSCATED C programs» [29]). Il mentionne explicitement le fait que lire le *code-source* est réellement nécessaire pour l'appréciation des codes présentés. De par ces éléments, on comprend que les codes mentionnés ne sont pas considérés aux yeux de l'auteur comme de simples codes, algorithmes ou programmes. Il en fait collection, parle d'appréciation et de préférence. En bref, tous ces éléments portent à croire que ces codes sont des œuvres d'art à ses yeux.

Bien qu'un seul contre-exemple soit suffisant pour monter l'absurdité de la proposition initiale, il n'est pas possible d'affirmer que le cas précédent est une preuve

irréfutable de contradiction. C'est plutôt la diversité des exemples et leur pertinence qui s'additionnent et finissent par former une preuve solide. Pour cela, on peut énumérer les exemples de l'IOCCC (7.1) ou les *codeworks* présentés en annexe (7.2). Tous ces exemples ont des spectateurs, des lecteurs et parfois même un jury pour attester de leur pertinence en termes de créativité. Ces derniers font naître des réactions visibles par les commentaires et les articles qui en découlent. Ils surprennent parfois (6), d'autres fois font preuve d'une grande complexité (4), peuvent être appréciés pour leur forme (5, 2), leur humour (7), parlent d'amour (12), font de la musique (10), de la poésie (9) ou racontent des histoires (11). Considérer la proposition initiale comme vrai implique de devoir considérer que tous ces éléments ne peuvent pas être considérés comme des manifestations de l'art. Cela rentre alors en contradiction avec les définitions discutées dans la partie sur la question de l'art (3).

Cette contradiction mise au jour montre que la proposition initiale ne peut être vrai. Nous avons donc démontré par l'absurde que la proposition initiale est fausse. On peut donc en conclure que l'on ne peut pas dire que le code n'est pas de l'art. Cela tend donc à montrer que le code en serait bien.

6.3 (Crêpe++)-- ou l'art de l'offuscation

Afin de présenter l'exemple d'un processus de programmation créative donnant lieu à un code pouvant être considéré comme de l'art, je vais maintenant présenter une expérience personnelle : la programmation d'un code intitulé *(Crêpe++)--* dans le langage C++.

J'ai ainsi participé à la 8ème édition de *MovaiCode*, un concours organisé par l'entreprise Coddity, et qui a lieu sur GitHub. L'objectif de cette édition spéciale Chandeleur était le suivant : «retirer le dernier élément d'une liste de strings» [60]. Ici, les chaînes de caractères sont censées représenter des crêpes. L'objectif initial est donc simple et sert avant-tout de prétexte à une démonstration de créativité de la part des candidats. Par exemple, certains candidats ont tenté de proposer la fonction avec le moins de caractères possible. En C++ avec les *lambda expressions*, la solution représente seulement 14 caractères : `[&]{a.p();}();`.

Mais je souhaitais faire quelque chose de plus créatif. En particulier, je venais de visionner sur une vidéo de la chaîne Bisqwit sur l'obfuscation de code en C [35] qui m'avait donné envie d'essayer cette forme de programmation créative. Cette vidéo explique bien les bases à garder en tête lors de la création de code obfusqué. Je me suis donné pour objectif de faire quelque chose d'effrayant et d'incompréhensible.

Puisqu'il faut afficher des `string`, j'ai créé un programme annexe afin de les rendre inintelligible. Le rendu final est un bloc de code relativement petit et qui contient à la fois les données sous forme des `string` et les instructions de manipulation de celles-ci. La démarche de création était réellement différente de celle que j'adopte d'habitude, et j'ai pris un réel plaisir à transgresser les règles et les principes traditionnels de la programmation.

Voici donc le code que j'ai proposé, intitulé *MovaiC++ pancakes : (Crêpe++)--, ou le côté obscur de la force* :

Code 1 – (Crêpe++)--

```
Crepe * MangerUneCrepe(Crepe * cpp) {
    if(cpp==nullptr)return nullptr;
    Crepe*l=cpp;Crepe*b=nullptr;
    while(l->next!=nullptr){b=l;
    l=l->next;}std::string p=l->crepe;
    std::vector<std::string>t={"rvbsd",
    "optmdu","ljdm","bbqblfk","kbqenor",
    "bi‘nojfonor","njfonor","splbsfr",
    "rpib","bvqsx","dyomnthg","ujsshpk",
    "rbupm","mfhhd","rbamd","bbhmkpty","omtunohvl"};
    auto L0=[&](std::string&str)
    {char buffer[str.length()];
    std::copy(str.begin(),str.end(),buffer);
    for(int i=0;i<(int)str.length();
    i++){(i%2==0)?buffer[i]++:buffer[i]--;}
    auto L1=[&](char* a,int size)
    {int i;std::string s="";for(i=0;i<size;i++)
    {s=s+a[i];}return s;};
    return L1(buffer,str.length());};auto L2=[&]()
    {return rand()%(t.size()-3)+3;};
    p+=L0(t[0])+"␣"+L0(t[L2()])+",␣"
    +L0(t[L2()])+",␣"+L0(t[1])+"␣"+ L0(t[L2()])
    +"␣"+L0(t[2])+"␣!";
    std::cout << p << std::endl; delete l;
    (b == nullptr) ? cpp = nullptr : b->next
    = nullptr;return cpp;
}
```

Afin de faire rentrer le code correctement dans ce document, j'ai dû modifier certaines lignes et la présentation de la fonction. Pour bien apprécier ce code, il peut être intéressant de connaître plus en détail le processus de création que j'ai suivi :

L'idée de base de ce code est assez simple. Il s'agit de construire une liste chaînée basique. Une fois la liste chaînée de `Crepe*` construite, on peut alors appeler la fonction `Crepe * MangerUneCrepe(Crepe * cpp)`. On sélectionne alors 4 ingrédients aléatoires sous forme de `std::string` afin de préparer la crêpe avant de la manger en affichant le message de la crêpe et de ses ingrédients. Ne reste qu'à

`delete` la dernière crêpe et à modifier le pointeur de la crêpe précédente (`nullptr`) (plus quelques cas de bord à gérer). Quand on appellera la fonction qui permet de manger une crêpe, on aura donc un message aléatoire de la forme :

```
crêpe lardons, soja, météorite supplément béton !
```

La liste des ingrédients est la suivante :

```
std::vector<std::string> crepeToppings = {
    "sucre",
    "chocolat",
        ...
    "meteorite",
    "regolite"
};
```

Comme je veux faire du code obfusqué, je ne veux pas qu'on puisse lire facilement le code, et encore moins les ingrédients, car ceux-ci sont parfois surprenants et je veux laisser la surprise intacte.

Je crée alors un petit programme `CLI obfuscator` qui va me permettre d'obfusquer simplement des strings (par défaut), mais aussi de les déobfusquer. On peut utiliser le programme directement en lignes de commandes :

```
(base) <onyr@kenzae> <obfuscator>> ./main hello world
<<<< You have entered 3 arguments >>>>
<<<< OBFUSCATE MODE >>>>

 + arg: hello
------> Obfuscated: gfkmn
------> Deobfuscated: hello
 + arg: world
------> Obfuscated: vpqmc
------> Deobfuscated: world
```

En passant le flag -d, le programme déobfusque les paramètres :

```
(base) <onyr@kenzae> <obfuscator>> ./main -d gfkmn vpqmc
<<<< You have entered 4 arguments >>>>
<<<< DE-OBFUSCATE MODE >>>>

    + arg: gfkmn
------> Deobfuscated: hello
```

```
-----> Obfuscated: gfkmn
    + arg: vpqmc
-----> Deobfuscated: world
-----> Obfuscated: vpqmc
```

La fonction `std::string obfuscateString(std::string & str)` est en fait très simple et va simplement décaler les caractères dans le tableau de `char` de la `std::string` fournie en paramètre.

```
// shift each char by +1 or -1 depending on its position
for (int i = 0; i < (int)str.length(); i++) {
    if (i % 2 == 0) {
        buffer[i]--;
    } else {
        buffer[i]++;
    }
}
```

Cet algorithme a le mérite d'être simple. En outre, il offre le double avantage d'être simplement réversible ainsi que de fournir des versions différentes de mêmes lettres, car le décalage dépend de la position de la lettre dans la `string`. On va donc pouvoir obfusquer notre `vector` de string avec ce petit programme. On va également ajouter les termes "crêpe", "supplément", "mangée" au début de notre `vector` afin de masquer ces mots de la même façon. Il faudra simplement veiller à ne pas pouvoir les sélectionner aléatoirement. On se retrouve avec une liste d'ingrédient incompréhensible que l'on devra *deobfuscate* avant d'afficher.

```
std::vector<std::string> crepeToppings = {
    "rvbsd",
    "optmdu",
    "ljdm",
    "bbqblfk",
        ...
    "bbqblfk",
    "njfonor"
};
```

Maintenant, il ne reste qu'à écrire la fonction
`Crepe * MangerUneCrepe(Crepe * c)` :

```
/**
 * @brief Remove the last crepe from the list
```

```
 *
 * @param crepes a list of crepes
 * @return Crepe* the remaining list of crepes
 */
Crepe * MangerUneCrepe(Crepe * crepes) {
    // return nullptr if no crepes
    if (crepes == nullptr) {
        return nullptr;
    }

    // get last and before last crepes
    Crepe * lastCrepe = crepes;
    Crepe * beforeLastCrepe = nullptr;

    while (lastCrepe->next != nullptr) {
        beforeLastCrepe = lastCrepe;
        lastCrepe = lastCrepe->next;
    }

    // Add topping to last crepe
    std::string lastCrepePrepared = lastCrepe->crepe;

    // add a random topping to the last crepe
    std::vector<std::string> crepeToppings = {
        "crepe",
        "supplement",
        "mangee",
        "sucre",
          ...
        "regolite"
    };

    // +x since we use the first x elements
    // of the vector differently
    int topping1 = (
        rand() % (crepeToppings.size() - 3)
    ) + 3;
    int topping2 = (
        rand() % (crepeToppings.size() - 3)
    ) + 3;
```

```
    int topping3 = (
        rand() % (crepeToppings.size() - 3)
    ) + 3;

    lastCrepePrepared += crepeToppings[0] +
        "␣" + crepeToppings[topping1] +
        ",␣" + crepeToppings[topping2] +
        ",␣" + crepeToppings[1] +
        "␣" + crepeToppings[topping3] +
        "␣" + crepeToppings[2] + "␣!";

    std::cout << lastCrepePrepared << std::endl;

    // remove the last crepe
    if (beforeLastCrepe == nullptr) {
        // if the last crepe is the only one, we delete it
        lastCrepe = nullptr;
    } else {
        // if the last crepe is not the only one,
        // we delete it
        beforeLastCrepe->next = nullptr;
    }

    return crepes;
}
```

On rentre alors dans la dernière partie, celle du processus de d'obfuscation manuelle. Pour cela, j'ai converti les fonctions externes, par exemple `std::string deobfuscateString(std::string & str)` en *lambda expressions* puisque le concours ne me permet (normalement) que de disposer d'une seule fonction.

Ci-dessous un exemple : la fonction `std::string convertToString(char* a, int size)` :

```
std::string convertToString(char* a, int size)
{
    int i;
    std::string s = "";
    for (i = 0; i < size; i++) {
        s = s + a[i];
```

```
    }
    return s;
}
```

Voici sa version convertie en *lambda expression* :

```
auto convert = [&](char* a, int size) {
    int i;
    std::string s = "";
    for (i = 0; i < size; i++) {
        s = s + a[i];
    }
    return s;
};
return convert(buffer, str.length());
```

On remplace également les structures `if` en *one-liners*. Une fois que toutes les fonctions externes à `Crepe * MangerUneCrepe(Crepe * c)` ont été convertis en lambdas et intégrées, on passe au remplacement des noms de variables et de fonctions. Puis on enlève les espaces inutiles dans le code et on compacte le tout en un gros bloc à même de décourager n'importe qui de lire le corps de la fonction. On obtient enfin le résultat souhaité.

Ce concours aura donc été l'occasion de découvrir une pratique différente de la programmation. Il est aussi très agréable de découvrir les propositions des autres participants. Comme on peut le lire dans les instructions du sujet : «ça semble facile de faire n'importe quoi mais finalement pas tant que ça.» [60]. En effet, il faut pouvoir accommoder ses envies créatives de la contrainte du code, mais l'expérience est intéressante et comme l'a dit un jour le cinéaste Orson Welles : «The enemy of art is the absence of limitation.» [62]. Finalement, j'ai vraiment eu l'impression de construire une sorte de poème obscur en jouant au programmeur fou. Il y avait une vraie sensation de création artistique, à transgresser certaines règles tout en devant en respecter d'autres. Enfin, la réception du code final et l'explication de la démarche ont également suscité des réactions. Une citation d'André Gide fera office de conclusion pour cette partie : «L'art naît de contraintes, vit de luttes et meurt de liberté.» [63].

7 Appendices

7.1 IOCCC codes

L'IOCCC, ou International Obfuscated C Code Contest est un concours annuel bien connu des programmeurs de C. Initialement créé par Landon Curt Noll et Larry Bassel en 1984, il s'agit d'un concours dans lequel l'objectif et de faire le programme le plus incompréhensible possible. Avec ses règles simples et la créativité des programmeurs, il est devenu un concours très populaire au fil des années. En outre, il serait le plus vieux concours d'internet toujours actif [51]. Parmi les propositions notables, en plus ce celles présentées ci-dessous, on mentionnera par exemple un programme composé de 0 octets, un système d'exploitation, un simulateur de vol, ou un jeu d'échec complet en quelques centaines de caractères seulement (voir l'article Wikipedia et le repo GihHub des gagnants) [50] [58].

Les codes et programmes ci-dessous proviennent du site de l'IOCCC (www.ioccc.org) et de la collection personnelle d'Enrique Bermúdez (www.cise.ufl.edu). Tous ces codes sont sous license *Creative Commons Attribution-ShareAlike 3.0 Unported License* : «Copyright (c) xxxx, Landon Curt Noll & Larry Bassel. All Rights Reserved. Permission for personal, educational or non-profit use is granted provided this copyright and notice are included in its entirety and remains unaltered. All other uses must receive prior permission in writing from both Landon Curt Noll and Larry Bassel. ». Pour plusieurs de ces programmes, l'intégrité du code n'a toujours pu être respectée pour des raisons d'affichage et de lisibilité. Il a parfois fallu changer légèrement la présentation ou enlever des lignes (replacées par [...]) afin de pouvoir tout afficher. Ainsi, il vaut mieux considérer les exemples présentés comme des illustrations quelque peu altérés des originaux. Ces exemples doivent servir d'entrée avant d'aller télécharger les codes présentés et bien d'autres encore sur les sites mentionnés. En outre, une part important du plaisir provient aussi de la découverte, de la compilation et de l'exécution de ces codes.

Célèbre code C en forme de donut ; par Andy Sloane (2006) :

Code 2 – Donut

```
k;double sin()
,cos();main(){float A=
0,B=0,i,j,z[1760];char b[
1760];printf("\x1b[2J");for(;;
){memset(b,32,1760);memset(z,0,7040)
;for(j=0;6.28>j;j+=0.07)for(i=0;6.28
>i;i+=0.02){float c=sin(i),d=cos(j),e=
sin(A),f=sin(j),g=cos(A),h=d+2,D=1/(c*
h*e+f*g+5),l=cos      (i),m=cos(B),n=s\
in(B),t=c*h*g-f*      e;int x=40+30*D*
(l*h*m-t*n),y=            12+15*D*(l*h*n
+t*m),o=x+80*y,          N=8*((f*e-c*d*g
)*m-c*d*e-f*g-l          *d*n);if(22>y&&
y>0&&x>0&&80>x&&D>z[o]){z[o]=D;;;b[o]=
".,-~:;=!*#$@"[N>0?N:0];}}/*#****!!-*/
 printf("\x1b[H");for(k=0;1761>k;k++)
  putchar(k%80?b[k]:10);A+=0.04;B+=
    0.02;}}/*****####*******!!=;:~
      ~::==!!!**********!!!==::-
        .,~~;;;========;;;:~-.
             ..,--------,*/
```

À compiler avec la commande `gcc -o donut donut.c -lm`. Nécessite un terminal ANSI ou VT100 pour fonctionner [52]. Ce code produit un donut qui tourne en 3D, le tout affiché via des caractères ASCII à la manière de pixels. Derrière ce programme, on trouve un calcul d'illumination en fonction de la rotation du tore, vie un calcul de projection. Des explications sur le code, ainsi que celui-ci en version non-obfusqué sont disponibles ici (medium.com) et là (a1k0n.net, notes de l'auteur). Il existe de nombreux clones de ce programme, ainsi qu'une version améliorée, baptisée *Donut Mark II* par Sloane (Embellishing the donut: an old-school CG cliche).

Un morceau de programme faisant usage du morse ; par Jim Hague (1986) :

Code 3 – Morse

```
#define DIT     (
#define DAH     )
#define __DAH   ++
#define DITDAH  *
#define DAHDIT  for
#define DIT_DAH malloc
#define DAH_DIT gets
#define _DAHDIT char
_DAHDIT _DAH_[]="ETIANMSURW[...]g7c8a90l?e'b.s;i,d:"
;main                       DIT                       DAH{_DAHDIT
[...]
DITDAH                      _DIT_,DITDAH              DIT_DAH DIT
DAH,DITDAH                  DAH_DIT DIT               DAH;DAHDIT
[...]
__DAH;_DIT==DAH_DIT         DIT _DIT                  DAH;__DIT
[...]
DAH_;__DIT                  DIT                       DITDAH
[...]
DIT'␣'DAH,DAH_ __DAH        DAH DAHDIT                DIT
[...]
DITDAH _DIT_!=DIT           DITDAH DAH_>='a'?         DITDAH
DAH_&223:DITDAH             DAH_ DAH DAH;             DIT
DITDAH                      DIT_ DAH __DAH,_DIT_      __DAH DAH
[...]
DAH;}_DAH DIT DIT_          DAH{                      __DIT DIT
[...]
```

Ce programme a été sélectionné pour la collection de t-shirt de 1987. Il a obtenu la mention *Worst abuse of the C preprocessor*. Note des juges : «Think Morse code when you ponder this program. Note how use of similar variables can be obfuscating! The author notes that this program implements the international Morse standard. Now for extra credit, what Morse message does the program spell out?» [32].

Un morceau du programme Babble, par Sean Barrett (1992). :

Code 4 – Babble

```
#define A(c,a,b) t=b;t+=a;_(c)
#define B(b,a) P(u,a)t=0;t-=a;R t+=u;t+=1;t/=2;_(b)
 [...]
#define Z(a,d) t=p;t-=d;t*=a;t+=d;_(p)r*=a;
#define _(a) t-=a;t*=r;a+=t;

V a,x,y,s C

                G(f(g),q)G(              g(h),2)n(B)Z(a,4)n(
          P)E(a,1)j(2989)H l k           (1)m(B)Z(a,6)l k(h(b))
       B(s,a                )m(P)            Z(s                ,8)E
     (a,q                     )Q(             s,h                (g)
    )Q(                                      f(c                 ),f
   (g)                                       )j(                 s)l
   i(b                                       (h)                 ,S,
  f(b                                        ))n                (B)
  E(a                                        ,2)              J(8)
l j(                                        x)Q(x,1)l n(P)Z(a,8
)o(x                                        ,g(f))d(f(e))J(5)l j
(x)l                                        d(x                )S(a
 ,x,                                        1)m                 (B)
[...]
    P)E(                      a,7)          l j                 (0)
      Q(s,1                 )l D(a          ,x,               q)o(
        a,q)S(a,h(g),a)i(s,                 M,a)i(a,A,g(f))i(q,D,x
            )o(s,16)B(a                     ,x)E(a,9)H l j(2766)H

T X(b(f))
```

D'après les notes même de l'auteur : «babble is a translator from a pseudo-assembly language into a subset of C suitable for execution on Charles Babbage's Analytical Engine. Or rather, the #defines in babble are that translator. The rest of babble is a babble program.» [31]. C'est un programme capable de faire des calculs de factorielles ou de plus grand facteur commun (PGCD) entre deux nombres. La forme du programme, CB, est une double référence, à Charles Baddage mais aussi à Charles Barrett : «babble was originally named 'cb', for obvious reasons, and is dedicated to the memory of Charles Barrett» [31].

Fragment d'un programme en palindrome ; par Merlyn LeRoy (Brian Westley), (1987) :

Code 5 – Palindrome

```
char rahc
[ ]
=
"\n/"
,
redivider
[ ]
=
"Able was I ere I saw elbA"
,
*
deliver ,reviled
=
1+1
,
niam ; main
( )
{/*\}
\*/
int tni
=
0x0
,
rahctup ,putchar
( )
,LACEDx0 = 0xDECAL,
rof ; for
(;(int) (tni);)
(int) (tni)
= reviled ; deliver =
redivider
```

«Every source code line is a palindrome!» [30]. Ce programme est un vrai palindrome ce qui est assez étonnant pour un programme en C. Il a logiquement obtenu la mention *Best Layout*. Note des juges : «Line by line symmetry performed better than any C beautifier. Think of if it as a C ink blot. :-)».

Un autre exemple de code C célèbre de l'IOCCC ; par Merlyn LeRoy (Brian Westley), (1988) :

Code 6 – Pi circle (westley 1988)

```
#define _ -F<00||--F-OO--;
int F=00,OO=00;main()
    {F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
                  _ _ _
                _ _ _ _
              _ _ _ _ _ _
            _ _ _ _ _ _ _ _
          _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _ _ _
      _ _ _ _ _ _ _ _ _ _ _ _ _ _
    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
      _ _ _ _ _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _ _ _
          _ _ _ _ _ _ _ _ _ _
            _ _ _ _ _ _ _ _
              _ _ _ _ _ _
                _ _ _ _
                  _ _ _
}
```

Ce code affiche `3.141` une fois compilé et exécuté. Malheureusement le code ne compile plus directement en C ANSI avec GCC sans modifications. On peut alors utiliser la version modifiée de Misha Dynin qui compile avec `gcc`. Voici le commentaire de Enrique Bermúdez : «A unique program layout and the most unusual method of calculating pi I have ever seen. The source code is a circle and the program works by calculating its own area and diameter, and then doing a division to approximate pi!! Very bizzare.» [30].

Morceau du programme *Pig-latin* ; par (Don Dodson), (1995) :

Code 7 – Piglatin

```
    X                                                       X
    X X                                                     X X
   X   X                                                   X   X
   X    X                                                  X    X
  X      X                                                 X      X
  X       X                                                X       X
 X         X                                              X         X
 X   X     X                                              X     X   X
 X   XX     X                                             X     XX  X
X    XXX    X         XXXXXXXXX                           X    XXX   X
X     XXX    X   XXXX          XXXX   X    XXX           X
X     XXXX   X XX ainma(){ archa  XX X   XXXX           X
X     XXXX    X   oink[9],*igpa,    X    XXXX           X
X     XXXXXX atinla=etcharga(),iocccwa XXXXXX           X
X      XXXX ,apca='A',owla='a',umna=26  XXXX            X
X      XXX  ; orfa(; (atinla+1)&&(!(((   XXX            X
X      XX atinla-apca)*(apca+umna-atinla) XX             X
 X     X  >=0)+((atinla-owla)*(owla+umna-  X            X
 X       atinla)>=0))); utcharpa(atinla),                X
 X   X atinla=etcharga()); orfa(; atinla+1;  X   X      X
  X X  ){ orfa(      igpa=oink      ,iocccwa=( X X
  X X  (atinla-  XXX  apca)*(  XXX apca+umna- X X
   X atinla)>=0) XXX            XXX   ; ((((    X
  X atinla-apca XXXXX XXXXXXX XXXXX  )*(apca+   X
  X umna-atinla XXXXXX )>=0) XXXXXX +((atinla-  X
 X owla)*(owla+ XXXX  umna-  XXXX atinla)>=0)) X
 X   &&"-Pig-"   XX  "Lat-in"   XX   "COb-fus"   X
 X "ca-tion!!"[  X  (((atinla-   X  apca)*(apca+ X
 X umna-atinla) X  >=0)?atinla-   X  apca+owla:  X
```

Ce programme clairement humoristique à la forme d'une tête de cochon et permet de convertir du texte anglais en langage cochon-latin. Il a logiquement obtenu la mention *Most Humorous*. Note de l'auteur : «The obfuscation is on several levels. Most obviously, the shape of the program. Underneath that, the variable names are in pig latin, as are the names of the standard C functions, such as putchar. Even main is written as ainma. The program construction is also very obfuscated, with all of the code being inside the ()'s of one of the 6 "orfa" loops.»

7.2 Codeworks

Les *codeworks* sont des œuvres littéraires et poétiques, c'est-à-dire des textes librement inspirés par les pratiques de la programmation et le code informatique. Bien qu'ils contiennent souvent des lignes de code ou sont empreints de notations typiques de certains langages de programmation, ceux-ci n'ont généralement pas vocation à être exécutés directement 4.2. Pour rappel : «Codework is a type of creative writing which in some way references or incorporates formal computer languages (C++, Perl, etc.) within the text. The text itself is not necessarily code that will compile or run, though some have added that requirement as a form of constraint.» [48]. Puisque ce terme reste un néologisme, il est possible d'utiliser d'autres expressions pour désigner cette nouvelle forme d'expression littéraire. On trouvera ainsi d'autres définitions, par exemple celle de Rita Raley dans son essai « Interferences : [Net.Writing] and the Practice of Codework » : «Codework refers to the use of the contemporary idiolect of the computer and computing processes in digital media experimental writing, or [net.writing].» [53]. Celle-ci fais la liste des termes alternatifs qui ont pu être utilisés :

— *codework* (œuvres), par Alan Sondheim
— *[net.writing]* (pratique), par Rita Raley
— *net.wurked language* (langage), par Mez (Mary-Anne Breeze)
— *rich.lit* (œuvres), par Talan Memmott
— *codepoetry* (œuvres), par Ted Warnell
— *digital visual poetics* (œuvres), par Brian Lennon
— *programmable poetry* (œuvres), par John Cayley

Toutes ces discussions et recherches font parties du mouvement *net.art* [53] [9]. Pour Rita Raley, tout cela fait partie d'un tout plus grand qui contribue à faire du code, un art : «Codework participates in a larger movement that we might call the "art of code," in which the code used to produce the work seems to infiltrate the surface, the former domain only of natural languages and numeric elements.» [53]. Malgré tout, on peut remarquer que tous ces néologismes sont en anglais, comme c'est souvent le cas des termes utilisés en informatique. En rapprochant les termes de code et de poésie, on pourrait proposer *codésie*, raccourci de la concaténation des deux termes, en remarquant que le code et poésie française partage une certaine idée de la contrainte, de la rigueur et de la transgression des règles. On peut conclure en citant une phrase de l'article, que « Le code est-il de la poésie ? » «Ceux qui rapprochent le code et la poésie affirment ainsi une certaine idée de ce que la technologie pourrait (ou devrait) être : joueuse, oblique, réflexive, parfois émouvante et critique.» [54].

Les *codeworks*, ou *codésies* ci-dessous sont des exemples concrets de comment

code, poésie et littérature peuvent se mêler.

Codésie en Perl valide intitulée *obfupoetry II*, par Discipulus (2019) :

Code 8 – Obfupoetry II

```
How:to:forget:when:I:was:young:
join '␣had␣', (glob'spr{a,e,i,o,u}ng')[2,4];
Now:I:get:in:the:middle:of:
join '␣and␣', (glob'Y{i,a}n{,g}')[0,-1];
and:I:remember:it:was:a:bang:
seek DATA,0,0;while(<DATA>){last if/DATA/;
chomp&&print/:/?(join$",split':',$_):eval,$/}
__DATA__
```

Note de l'auteur : «In occasion of first day of Spring* i give you the second issue of perl obfupoetry. (*)please do not reply with positivist comments about the exact first day of Spring: seasons always start in day 21: the notion is in my mind since...» [55].

Codésie en C++ valide, intitulée *C++ Haiku*, par tvaneerd (2017) :

Code 9 – C++ Haiku

```
// classic example
using namespace std;
cout << "Hello,␣world!";
```

Ce poème respecte la structure d'un Haiku. Note de l'auteur : «pronunciation of cout and std to get 5/7/5 is left as an exercise for the reader». Ici, on prononcera donc `std` lettre-par-lettre, et `cout` comme «C out» [59].

Codésie en Perl valide intitulée *12 days of Perl ?*, par johnaj (2021) :

Code 10 – 12 days of Perl ?

```
for(<DATA>){s/%/ing/;$_=($i++?$i-1+print"
":a)."␣$_";print"On␣the␣$i",(st,nd,rd)[$i-1]||th,
"␣day␣of␣Christmas,
my␣true␣love␣gave␣to␣me:
",$t=$_.$t}__DATA__
partridge in a pear tree.
turtle doves and
French hens,
call% birds,
golden rings,
geese a-lay%,
swans a-swimm%,
maids a-milk%,
ladies danc%,
lords a-leap%,
pipers pip%,
drummers drumm%,
```

Note de l'auteur : «I've been working on an exercise to generate the English "Twelve days of Christmas song"» [56].

Fragment du *Internet Text*, par Alan Sondheim :

Code 11 – Internet Text, fragment de net3.txt

```
You transform each and every representation,
    by virtue of protocol,
camouflage, or terminal emulation.
    Every screen act is one of
mimickry; your fears and desires,
    which are identical, reside within
it. ASCII text always aligns itself with the rectangular...

                        /
                x                       / z
    ___________________________________/

    |                                   |

    |                                   |

    |                                   | y

    |                                   |

    |___________________________________|

z is a particular construct, the thrust or
    interpenetration of the
screen, the simulacrum of the breast or ultimate
    coordination which
cannot be surpassed. Just as the horizon of
    Greenland up to the
Central Ridge is always tilted, white against white,
    a blindness
sliding the uncanny, so the screen is almost always
    angled against
the real, even if perpendicular default.
    The real WEDGES ITSELF.
The real TURNS AWAY FROM YOUR EYE.
```

Issue du fichier `/1/works/sondheim__internet_text/net3.txt` [57]. Le découpage des lignes a dû être modifié pour rentrer correctement dans ce document.

Fragment de codésie en C valide intitulée *She loves me, she loves me not, ...*, par Merlyn LeRoy pour l'IOCCC (1990) :

Code 12 – Charlie and Charlotte

```
char*lie;

    double time, me= !0XFACE,

    not; int rested,   get, out;

    main(ly, die) char ly, **die ;{

        signed char lotte,


dear; (char)lotte--;

    for(get= !me;; not){

    1 -  out & out ;lie;{

    char lotte, my= dear,

    **let= !!me *!not+ ++die;

        (char*)(lie=
```

Tiré de la collection d'Enrique Bermúdez. Voici son commentaire sur ce code : «You must see the source code layout to appreciate this; it reads just like a series of letters exchanged between charlie and charlotte. This is extraordinary!» [30].

Références

[1] Alan Turing. « On computable numbers, with an application to the entscheidungsproblem ». In : *Proc. Lond. Math. Soc. (3)* s2-42.1 (1937), p. 230-265.

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

[3] M Romero, B Lille et A Patiño. *Usages créatifs du numérique pour l'appren-tissage au XXIe siècle*. Presses de l'Université du Québec, 2017.

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

[5] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976, p. vii-30.

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

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

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

[9] Florian Cramer. « Digital code and literary text ». In : *Beehive Hypertext/-Hypermedia Literary Journal* (27 sept. 2001).

[14] René Magritte. *La trahison des images (Ceci n'est pas une pipe)*. Paintings : Oil on canvas 23 3/4 × 31 15/16 × 1 in. Currently on public view : Broad Contemporary Art Museum, floor 3. 1929.

[15] Simon Davies. *Definitions of Art*. Cornell University Press, 1991. isbn : 978-0-8014-9794-0.

[16] Donald E. Knuth. *The art of computer programming*. 3rd Ed. T. 1. Addison-Wesley Longman, 1997. 664 p. isbn : 0-201-89683-4.

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

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

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

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

[21] B Jack Copeland. « The modern history of computing ». In : *The Stanford Encyclopedia of Philosophy*. Sous la dir. d'Edward N Zalta. Winter 2020. Metaphysics Research Lab, Stanford University, 2020.

[22] Joasia Krysa. « Ada Lovelace : There Never Was a Note G ». In : (). Sous la dir. de 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.

[23] Arthur Danto. *La transfiguration du banal : une philosophie de l'art*. Edition du Seuil, 1989. isbn : 978-2-02-010463-0.

[24] Marcus Steinweg. « Nine Theses on Art ». In : *Art U+0026 Research* 3.1 (2009). issn : 1752-6388.

[25] *art / Wiktionnaire*. fr. Accessed : 2022-5-12.

[26] Éditions Larousse. *Définitions : oeuvre - Dictionnaire de français Larousse*. fr. Accessed : 2022-5-12.

[27] Editions Larousse. *Définitions : art, arts - Dictionnaire de français Larousse*. fr. Accessed : 2022-5-12.

[28] Dean Takahashi. *Jonathan Blow : How Thekla is moving beyond The Witness*. en. Accessed : 2022-5-13. Juill. 2018.

[29] Manuel Enrique Bermúdez. *Manuel Enrique Bermúdez THEN ... and NOW ...* Accessed : 2022-5-17. url : `https://www.cise.ufl.edu/~manuel/`.

[30] Manuel Enrique Bermúdez. *Obfuscated C Code*. Accessed : 2022-5-17. url : `https://www.cise.ufl.edu/~manuel/obfuscate/obfuscate.html`.

[31] Sean Barrett. *babble*. Accessed : 2022-5-13.

[32] Jim Hague. Accessed : 2022-5-13.

[33] Don Dodson. Accessed : 2022-5-13.

[34] Francois Theurel. *LE DERNIER VRAI NANAR*. Oct. 2020. url : `https://youtu.be/RCSiFM_ZLP8`.

[35] Bisqwit. *Obfuscated C programs : Introduction*. Fév. 2016. url : `https://youtu.be/rwOI1biZeD8`.

[36] *Filmspotting, scariest movies, film, podcast, reviews, DVDs, Adam kempenaar*. en. Accessed : 2022-5-16.

[37] Little White Lies magazine. *Cannibal Holocaust*. en. Accessed : 2022-5-16.

[38] Tom Cowie. « If it's anywhere, it's here' : The hunt for a cult film lost 50 years ago ». en. In : *Age* (mai 2022).

[39] Didier Müller. « Les différents types de démonstrations ». In : (). Accessed : 2022-5-16.

[40] Sylvie CALABRETTO. « Approche logique de l'Intelligence Artificielle ». In : (2021). Cours INSA Lyon, 4IF (2021-2022).

[42] Peter KIVY. « Is Music an Art ? » In : *Journal of Philosophy* 88.10 (1991), p. 544-554.

[43] Philip DORRELL. *What is Music ?* 2005. Chap. 2.

[44] Wikipedia CONTRIBUTORS. *Music*. Accessed : 2022-5-17. Mai 2022. URL : https://en.wikipedia.org/w/index.php?title=Music&oldid=1088431682.

[45] WIKIPEDIA CONTRIBUTORS. *List of failed and overbudget custom software projects*. Accessed : 2022-5-23. Mars 2022. URL : https://en.wikipedia.org/w/index.php?title=List_of_failed_and_overbudget_custom_software_projects&oldid=1078214926.

[46] Yaneer BAR-YAM. « When systems engineering fails — Toward complex systems engineering ». In : 2 (2003). Accessed : 2022-5-23.

[47] Gabi DOBOCAN. *Tech fails : A big bet that cost the IRS billions - north code - medium*. en. Accessed : 2022-5-23. Fév. 2020.

[48] *Contents by Keyword*. Oct. 2006.

[49] Kenneth IVERSON. « Notation as a Tool of Thought ». In : (1979). 1979 ACM Turing Award Lecture.

[50] Wikipedia CONTRIBUTORS. *International Obfuscated C Code Contest*. Accessed : 2022-5-28. 2022. URL : https://en.wikipedia.org/wiki/International_Obfuscated_C_Code_Contest.

[51] Landon Curt Noll ; Volker Diels-Grabsch ; Yusuke Endoh ; Kang SEONGHOON. *About the IOCCC winner GitHub repository*. 2022. URL : https://github.com/ioccc-src/winner.

[52] Andy SLOANE. *Have a donut*. Accessed : 2022-5-28. Sept. 2006. URL : https://www.a1k0n.net/2006/09/15/obfuscated-c-donut.html.

[53] Rita RALEY. « Interferences : [Net.Writing] and the Practice of Codework ». In : (août 2002). Accessed : 2022-5-29.

[54] Claire RICHARD. « Le code est-il de la poésie ? » In : (nov. 2016). Accessed : 2022-5-29.

[55] JOHNAJ. *12 days of Perl ?* Accessed : 2022-5-29. 2021. URL : https://www.perlmonks.org/?node_id=1231522.

[56] JOHNAJ. *12 days of Perl ?* Accessed : 2022-5-29. 2021. URL : https://www.perlmonks.org/?node_id=11131906.

[57] Alan SONHEIM. *Internet Text*. Accessed : 2022-5-29. URL : https://collection.eliterature.org/1/works/sondheim__internet_text/mq.txt.

[58] Landon Curt BROUKHIS Leonid A ; Noll. *International Obfuscated C Code Contest*. Accessed : 2022-5-28. URL : https://www.ioccc.org/.

[59] TVANEERD. *A c++ poem*. Accessed : 2022-5-29. URL : https://www.reddit.com/r/cpp/comments/7crald/a_c_poem/.

[60] LOUISMARSLEN. *MOVAI CODE 8 - CHANDELEUR, go manger des crêpes*. Accessed : 2022-5-29. URL : https://github.com/CoddityTeam/movaicode/blob/master/saison-2021-2022/0222/Movaicode-8-2022-02.md.

[61] Wikipedia CONTRIBUTORS. *Category :Films about virtual reality*. Accessed : 2022-5-31. URL : https://en.wikipedia.org/wiki/Category:Films_about_virtual_reality.

[62] Noam KROLL. *'The Enemy Of Art Is The Absence Of Limitations' and How Boxing Yourself In Will Help You Develop The Best Story Idea*. Accessed : 2022-5-30. URL : https://noamkroll.com/the-enemy-of-art-is-the-absence-of-limitations-how-boxing-yourself-in-will-help-you-develop-the-best-story-idea/.

[63] *L'art naît de contraintes, vit de luttes et meurt de liberté*. URL : http://evene.lefigaro.fr/citation/art-nait-contraintes-vit-luttes-meurt-liberte-19935.php.

Bibliographie complémentaire

[10] Andy ORAM et Grew WILSON. *L'art du beau code*. 1ère Ed. Paris : Edition O'REILLY, 2008. 621 p. ISBN : 978-2-84177-423-4.

[11] Martin FOWLER. *Refactoring*. 2nd Ed. Malakoff : Dunod, 2019. 419 p. ISBN : 978-2-10-080116-9.

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

[13] Dustin BOSWELL et Trevor FOUCHER. *The art of readable code*. 1st Ed. O'Reilly Media, Inc, 2011. 204 p. ISBN : 978-0596802295.

[41] WIKIPEDIA CONTRIBUTORS. *Modus ponens*. Accessed : 2022-5-17. URL : https://fr.wikipedia.org/w/index.php?title=Modus_ponens&oldid=193537939.