Initier les élèves à la pensée informatique et à la programmation avec Scratch

Version du 20 janvier 2016

Pierre Tchounikine

Université Grenoble-Alpes - Pierre.Tchounikine@imag.fr

Introduction

L'enseignement de l'informatique, de façon générale et, notamment, à l'école élémentaire, est actuellement un objet de réflexion.

Il y a sur le sujet des discours très divers, plus ou moins fondés, et également plus ou moins liés aux enjeux sous-jacents (disciplinaires, médiatiques, politiques). Il y a beaucoup d'appels en faveur de cet enseignement, avec des arguments différents, étayés ou pas ; en substance : « cela permet de développer des compétences importantes », « cela développe la créativité des élèves », « c'est moderne », « c'est un domaine scientifique important », « c'est un secteur qui embauche », « mon petit frère adore », ou encore « c'est mon domaine d'activité, donc il est important, donc il faut l'enseigner ». Il y a également beaucoup de confusions (par exemple qu'enseigner l'informatique a pour but de faire des élèves de futurs programmeurs). Il y a enfin des ressources pédagogique mais qui, si l'on n'a pas les idées claires sur ce que l'on veut enseigner, pourquoi et comment, peuvent être difficiles à exploiter.

L'objectif de ce document est d'essayer d'aider les enseignants (et les formateurs d'enseignants) à y voir plus clair. Pour cela, il propose des informations et des réflexions pour connaitre et comprendre le domaine, les discours, les programmes et les principales ressources existantes mais, surtout, pour développer une réflexion personnelle sur quoi enseigner, pourquoi et comment.

Le contenu porte sur deux aspects, l'enseignement de « la pensée informatique » et l'utilisation de l'environnement de programmation Scratch. Aborder l'informatique via la notion de « pensée informatique » m'amène à mettre au cœur de la réflexion et de l'enseignement l'algorithmique et la programmation, dans une approche traditionnelle ou orientée « informatique créative ». Ceci ne résume évidemment pas la discipline informatique, et d'autres approches (mettant plus en avant d'autres notions : machine, automate, langage, bases de données, génie logiciel, architecture, composants électroniques, etc.) sont possibles. Je fais ce choix car c'est celui qui me semble le plus approprié pour une initiation à l'école et, par ailleurs, qu'il est cohérent avec ce que l'on trouve dans les programmes de l'éducation nationale. Dans la suite du document, lorsque j'utilise le terme « informatique » pour faire plus court, il s'agit donc de cette approche de l'informatique. Par souci de simplification pédagogique, je prends par ailleurs par moment un peu de liberté avec les canons de la discipline (quelques approximations qui n'induiront pas de mauvaises conceptions je l'espère). Le support technique que j'utilise est le langage Scratch. Pourquoi Scratch ? Parce que c'est le langage qui s'impose de fait pour le niveau de l'école primaire, et qu'il y a de bonnes raisons à cela. L'essentiel du discours est cependant indépendant de ce langage particulier[1].

Le texte est principalement écrit pour des professeurs des écoles (instituteurs et institutrices)[2]. Il s'adresse donc à des lecteurs qui, pour la plupart, n'ont pas suivi de formation à l'informatique[3] au cours de leurs études. Le but n'est pas de les former à l'informatique (ce n'est pas un cours d'informatique), mais de donner des éléments permettant de comprendre ce que peut être un enseignement de la « pensée informatique » et comment l'aborder. C'est un point d'entrée donc. Pour se lancer dans un enseignement effectif il faut ensuite développer une certaine maitrise de ce que

[1] Il existe beaucoup d'autres langages de programmation ou environnements informatiques pour élèves/enfants, ainsi que des dispositifs de construction/pilotage de robots, des ateliers de composants électroniques, etc. Dans ce document je me limite à indiquer comment utiliser Scratch car c'est la référence, c'est simple et c'est gratuit (ce qui ne veut pas dire que c'est le mieux pour tout bien sûr). Les passionnés sauront trouver et exploiter d'autres ressources.

[2] Ce document a été rédigé dans le cadre d'un enseignement à l'Espé (École supérieure du professorat et de l'éducation) de Grenoble.

[3] Je parle ici de l'informatique comme discipline, à ne pas confondre avec les formations visant à apprendre à se servir d'un ordinateur et des logiciels standards type bureautique, moteurs de recherche, etc.

l'on veut enseigner (comme on le verra, c'est abordable quelle que soit sa discipline d'origine, et ce document est presque suffisant pour cela), puis exploiter les ressources actuellement à disposition (le document présente des exemples réutilisables, pas d'activités/séquences prêtes à emploi ; cependant, des pointeurs vers ce type de ressources sont indiqués).

Etant donné le public cible, le niveau considéré est le cycle 3 (école / CM1 et CM2). Il est cependant possible de faire des choses au cycle 2 et, par ailleurs, certains éléments sont également potentiellement pertinents pour les niveaux collège/lycée (Scratch reste pertinent pour le collège, voire le lycée pour certains aspects ; à ces niveaux, il est possible d'utiliser d'autres langages de programmation, mais les réflexions pédagogiques sont similaires).

Le texte est structuré autour de la liste de questions suivantes :

1. Qu'est-ce que c'est que la « pensée informatique » ? (définitions, exemples, confusions à éviter).
2. Pourquoi enseigner l'informatique à l'école primaire ? (arguments, programmes actuels, position personnelle).
3. Que faut-il faire pour enseigner l'informatique à l'école ?
4. Quels objectifs pédagogiques peut-on considérer ? (différents objectifs possibles puis discussion autour des conceptions/approches orientées « algorithmique » et « informatique créative » de l'enseignement de l'informatique).
5. Que veut dire « utiliser Scratch pour enseigner la pensée informatique » ? (principes généraux et différents exemples pour comprendre la différence entre *utiliser* et *faire* un algorithme/programme, comment aligner le moyen Scratch et l'analyse pédagogique/didactique, comment utiliser Scratch pour scénariser une situation pédagogique ou comment passer de la production de texte à l'informatique).
6. Que faut-il comprendre de Scratch en tant qu'enseignant, et comment s'y prendre ?
7. Comment définir et gérer des situations pédagogiques ?
8. Quelles difficultés peut-on attendre/anticiper ?

Le texte a par ailleurs plusieurs niveaux de lecture : il est informatif et, sur certains sujets, aborde des questions plus fondamentales. Bien que les premières questions abordées ne soient pas liées à un langage particulier, des exemples Scratch sont présentés dès le début du texte afin de faciliter la compréhension.

A propos de l'auteur, afin de mieux comprendre le discours développé : je suis Professeur d'informatique à l'Université. J'enseigne donc l'informatique (algorithmique, langages divers, génie logiciel, conception orientée objet, etc.) à des étudiants en sciences et/ou en informatique. Je mène par ailleurs des travaux de recherche sur des questions relatives à la conception d'environnements informatiques pour l'enseignement et l'apprentissage, notamment sur les questions d'apprentissage collaboratif[4]. Cette activité de recherche m'amène à travailler avec des cadres théoriques issus des Sciences Humaines et Sociales, des chercheurs en éducation et, bien sûr, des enseignants « de terrain », à l'école élémentaire tout particulièrement.

J'espère pouvoir faire évoluer ce texte et remercie par avance les lecteurs pour leurs commentaires[5].

1. Qu'est-ce que c'est que la « pensée informatique » ?

1.1. Définitions des notions de pensée informatique et d'algorithme

Le terme « pensée informatique » est généralement utilisé dans le sens de : notions et façons de faire que l'on trouve de façon explicite et prégnante en informatique, notamment la notion d'algorithme. On pourrait évidemment discuter de ce terme (est-ce une forme de pensée, est-elle vraiment liée à l'informatique, de quoi se différencie-t-elle, etc.), mais ce n'est pas l'objet de ce document.

[4] Pour les lecteurs intéressés, consulter http://lig-membres.imag.fr/tchounikine.

[5] Le texte actuel a notamment bénéficié de commentaires de (par ordre alphabétique) : Gwladys Agussol (professeur des écoles), Audrey Arnaud (étudiante M2 Espé Grenoble), Denis Bouhineau (enseignant/chercheur en informatique, Grenoble), Hamid Chaachoua (enseignant/chercheur en didactique des maths, Espé Grenoble), Nathalie Clouvel (professeur des écoles), Candice Delage (professeur des écoles), Sébastien Jolivet (formateur numérique, Espé Grenoble), André Tricot (enseignant/chercheur en psychologie, Espé Toulouse). Ce qui n'engage pas leur responsabilité dans le texte final bien sûr.

Le concept d'algorithme est au cœur de la pensée informatique. Avant de continuer sur cette idée de « pensée informatique » je fais donc ici une synthèse sommaire sur la notion d'algorithme (tout ceci sera exemplifié en Section 1.2). Un algorithme est un enchainement mécanique d'actions, dans un certain ordre, qui chacune a un effet, et dont l'exécution complète permet de résoudre un problème ou de faire quelque chose. La notion d'algorithme n'est pas propre à l'informatique : en maths le programme de cycle 2 indique la compétence cible « Mettre en œuvre un algorithme de calcul posé pour l'addition, la soustraction, la multiplication ». Une recette de cuisine est un algorithme, dont les actions sont de « casser les œufs », de « mettre de la farine » ou de « si nécessaire, rajouter un peu de sel ». En informatique, les actions dont il est question sont, par exemple, d'afficher une information à l'écran, de demander une donnée à l'utilisateur ou de faire un calcul. Elles sont souvent structurées par des boucles (qui permettent de répéter une séquence d'actions) et/ou des structures conditionnelles (structures Si-Alors-Sinon, qui permettent de faire une séquence d'actions ou une autre en fonction de l'évaluation d'une condition). Un programme est un algorithme traduit dans un langage de programmation. Un ordinateur est une machine qui (entre autres) est capable d'exécuter un programme, i.e., de réaliser les actions du programme / algorithme.

Le terme « pensée informatique » a fait florès depuis la parution d'un article « point de vue » d'une chercheuse en informatique aux Etats Unis[6] où elle argumente que la pensée informatique est « un ensemble d'attitudes et de connaissances universellement applicables, que nous gagnerions toutes et tous à apprendre et à maîtriser, et que nous devrions transmettre à nos enfants ». Il y a toute une littérature proposant des définitions plus précises[7], dont celle proposée par cette même chercheuse en 2011 : «Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent." Au niveau Français un article souvent cité[8] présente les choses ainsi : « [Le] souci du caractère algorithmique de la description des objets, du langage dans lequel ces descriptions sont exprimées, des flux d'information et des instruments sont plus généralement caractéristiques d'une « pensée informatique » ». On retrouve les mêmes idées dans les textes (co-signés par différents acteurs plus ou moins institutionnels) demandant de prise en compte de l'enseignement de l'informatique dans les programmes du primaire et du secondaire, cf. Section 2).

La définition à laquelle font référence les promoteurs de l'enseignement de la pensée informatique à l'aide du langage Scratch est intéressante à double titre. D'une part, elle détaille les choses. D'autre part, elle permet de comprendre la vision sous-jacente aux ressources pédagogiques proposées autour de Scratch. Cette définition est[9] :

- « [La pensée informatique] implique d'appréhender le monde selon l'approche employée en programmation par les développeurs de logiciels.
- Cette approche peut être scindée en cinq grandes catégories :
  - appréhender un problème et sa solution à différents niveaux (abstraction) ;
  - réfléchir aux tâches à accomplir sous forme d'une série d'étapes (algorithmes) ;
  - comprendre que pour résoudre un problème complexe il faut le décomposer en plusieurs problèmes simples (décomposition) ;
  - comprendre qu'il est probable qu'un nouveau problème soit lié à d'autres problèmes déjà résolus par l'élève (reconnaissance de formes), et
  - réaliser que la solution à un problème peut servir à résoudre tout un éventail de problèmes semblables (généralisation) ».

La pensée informatique ne se réduit donc pas à l'algorithmique mais fait référence à une façon d'aborder les problèmes qui va en général conduire à l'écriture d'un algorithme en langage naturel plus ou moins structuré et, éventuellement, d'un programme, i.e., une traduction de l'algorithme dans un langage exécutable par une machine.

Sur la base de ces définitions, le contenu de ce document peut être précisé : c'est une réflexion sur un enseignement qui amène les élèves à travailler sur des algorithmes et à les programmer, en l'occurrence en Scratch. Comme on le verra, ceci peut cependant correspondre à des objectifs pédagogiques et des réalités très différentes.

---

[6] Jeannette Wing, La pensée informatique (traduction, 2006) ; https://interstices.info/jcms/c_43267/la-pensee-informatique.

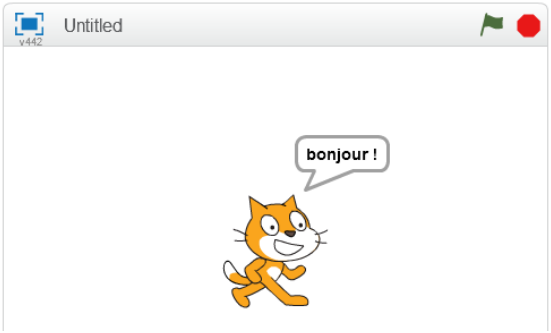
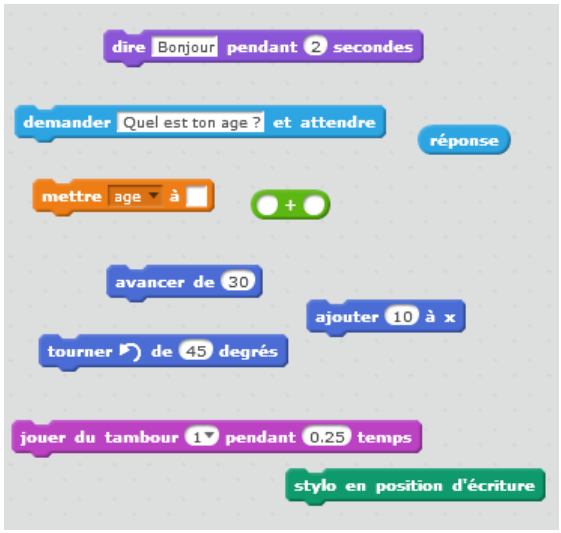
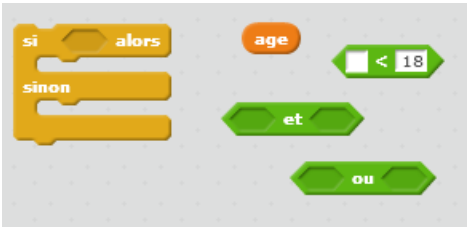
[7] Pour une revue de définitions dans un contexte d'enseignement voir par exemple : Grover S., Pea R. (2013). Computational thinking in K-12: A review of the state of the field. Educational Researcher, 42 (2), 59–69.

[8] Gilles Dowek, Les quatre concepts de l'informatique (2011) ; http://www.epi.asso.fr/revue/articles/a1204g.htm.

[9] Cf. les parties introductives du livre « Bien commencer avec Scratch, introduction à l'informatique » (traduction, 2013) ; https://pixees.fr/?p=3372

1.2. Algorithme et programmation : de quoi parle-t-on exactement ?[10]

Afin de fixer les idées et de montrer dès le début de ce texte que l'on parle de choses simples et accessibles, la Figure 1 montre des exemples d'actions et de structures algorithmiques[11] et de leur représentation dans le langage Scratch[12].

	En Scratch un programme est une série d'actions associées à un personnage[13] qui évolue sur une scène. Par exemple ici, le personnage effectue l'action : « dire « bonjour » ». Un programme peut être lancé en cliquant sur le drapeau vert et/ou en cliquant sur le personnage.
	Exemples d'actions : afficher un message ou une information ; demander une valeur (elle sera récupérée dans « réponse ») ; donner une valeur à une variable[14] ; faire des opérations (addition, multiplication, etc.) ; faire bouger un personnage (avancer, tourner, aller à une position -etc.-, ce qui peut se faire en indiquant les valeurs de sa position par les variables prédéfinies x et y, qui correspondent aux coordonnées du personnage sur la scène) ; jouer un son ; faire en sorte que quand le personnage se déplace il laisse une trace (mettre le stylo en position « écriture »). Séquence d'actions = lier plusieurs actions (en rapprochant les blocs, qui vont s'encastrer façon puzzle), par exemple « dire bonjour ; demander l'âge ; transférer la valeur dans la variable « age »).
	Boucle : répéter (ici, 10 fois) les actions qui seront mises dans le bloc.
	Structure conditionnelle : en fonction de l'évaluation de la condition qui sera mise entre le « Si » et le « Alors » (par exemple, « age < 18 »), faire les actions qui seront dans la première partie du bloc (la partie « Alors ») ou dans la seconde partie du bloc (la partie « Sinon »). La condition pourrait être multiple, par exemple en utilisant un bloc « et » ou un bloc « ou ».

[10] En situation d'enseignement présentiel il ne faut pas présenter/regarder ce tableau mais présenter/se familiariser avec ces éléments via une démonstration/manipulation sur machine.

[11] Un point de vocabulaire : les notions ne sont pas strictement identiques mais ce qui est appelé « action » ici correspond en général à ce qui est appelé « instruction » dans les langages de programmation et « bloc » en Scratch.

[12] Pour ceux qui découvrent l'algorithmique et la programmation Scratch : ce qui est dans ce tableau est déjà plus que ce qu'il faut avoir compris pour travailler avec des élèves de cycle 3 (la notion de variable n'est pas obligatoire) ; cf. Section 3.2.

[13] Cette notion de personnage n'est pas une notion d'algorithmique, elle est liée de l'approche Scratch.

[14] En informatique une variable peut être vue comme une « boite » dans laquelle on met une valeur. En Scratch, quand on demande à l'utilisateur de taper quelque chose (par exemple, son âge), la valeur ou le texte qu'il tape vont être mis dans la boite/variable « réponse » ; on peut ensuite transférer cette valeur dans une autre boite/variable à laquelle on donne un nom, par exemple le nom « age ».

Un programme (ou « script » en terminologie Scratch) associé à un personnage (le chat). Quand on cliquera sur le personnage, la série d'actions suivante va être réalisée : dire « bonjour » ; demander l'âge ; mettre la « réponse » (i.e., la valeur tapée au clavier par l'utilisateur) dans la variable « age » ; si la valeur est inférieure à 18 (premier cas de la conditionnelle Si-Alors-Sinon), répéter 8 fois « faire avancer le personnage de 30 pas et le faire tourner de 45° » (il va faire un joli saut périlleux), puis dire « salut mon gars ; » si la valeur de la variable age est supérieure à 18 (deuxième cas de la conditionnelle Si-Alors-Sinon), alors dire plus sobrement « bonjour Monsieur ».

A noter : la séquence d'actions permettant de demander et récupérer des informations via une question à l'utilisateur aura en général toujours la même structure (question + saisie réponse) ; elle peut être vue comme une solution abstraite réutilisable dans d'autres programmes.

1.3. Confusions et mauvaises interprétations à éviter

L'informatique est un domaine qui est souvent confondu avec ses applications, notamment car tout le monde utilise ses supports techniques (ordinateur, tablette, téléphone) et ses applications (éditeurs de texte, tableaux intégrant des formules de calculs, navigateurs Internet, logiciels de retouche photo, etc.).

Lorsque l'on enseigne l'informatique, il faut faire tout particulièrement attention à :

- Ne pas confondre l'enseignement de l'informatique (i.e., typiquement, amener les élèves à faire des algorithmes et des programmes) avec l'enseignement de l'usage d'un ordinateur (i.e., typiquement, apprendre à utiliser clavier/souris et un système d'exploitation comme Windows) ou de l'usage de logiciels standards (éditeurs de textes, etc.) ; ces compétences-là sont l'objet de formations type B2i.
- Ne pas confondre l'enseignement de notions relevant de l'informatique et l'utilisation de l'informatique pour enseigner (par exemple, l'utilisation de logiciels de géométrie dynamique pour enseigner les maths ou l'utilisation de TBI pour enseigner l'histoire). Même si, bien sûr, on peut utiliser l'informatique pour enseigner l'informatique, et c'est notamment ce que l'on fait quand on utilise la programmation pour enseigner l'algorithmique.

Un point très important est qu'il ne faut donc pas voir l'enseignement de l'informatique comme lié à une suite de niveaux de compétences de l'enseignant qui seraient : niveau 1, je sais enseigner aux élèves l'utilisation d'un ordinateur (grosso modo, la bureautique) ; niveau 2, je sais exploiter un TBI, des simulations à caractères pédagogiques, les logiciels pour la géométrie dynamique, etc. ; niveau 3, je sais enseigner l'informatique et la programmation. Enseigner l'informatique n'est pas de même nature et n'a rien à voir avec l'utilisation pédagogique de ressources pédagogiques informatisées ou d'un TBI, si ce n'est que l'on utilise un même support : un ordinateur (mais on utilise aussi un même support, le tableau, pour enseigner les maths et le français). Il est tout à fait possible d'enseigner la pensée informatique sans être spécialiste de bureautique, des TBIs ou de logiciels de géométrie dynamique (il faut des compétences, mais pas celles-là).

Par ailleurs, la pensée informatique telle que décrite ci-dessus et les compétences sous-jacentes ne sont pas intrinsèques ni même indissociablement liées à la notion d'ordinateur. « Penser » en terme d'algorithme ne nécessite absolument pas d'ordinateur, et on peut faire de l'informatique, et même de la programmation, sans ordinateur (informatique « débranchée » ou « unplugged »)[15]. Le lien avec l'ordinateur est qu'un ordinateur est un exécuteur

[15] A titre d'exemple, un exercice classique d'informatique « débranchée » est de simuler un algorithme/programme de pilotage d'un robot. Le but est de faire circuler un robot d'un point de départ à un point d'arrivée, dans la classe ou dans la cour d'école. Les élèves doivent écrire un algorithme enchainant une séquence d'actions comme « avancer de 1 pas », « tourner droite », « tourner gauche », etc. Le robot est remplacé par un élève et l'ordinateur est remplacé par les autres élèves, qui appliquent l'algorithme en indiquant à leur camarade les actions prévues. Je ne détaille pas plus cette approche de l'enseignement de l'informatique, qui est une alternative et/ou un complément à l'approche « sur machine ». Pour des ressources sur cette approche voir par exemple « L'informatique sans ordinateur, programme d'activités d'éveil pour les élèves à partir de l'école primaire » (https://interstices.info/upload/docs/application/pdf/2014-06/csunplugged2014-fr.pdf) ou encore le site « Informatique sans Ordinateur » de l'Irem de Clermont Ferrand (http://www.irem.univ-bpclermont.fr/Informatique-sans-Ordinateur).

d'algorithmes / de programmes. Construire des programmes amène donc à travailler sur (réfléchir à, construire, analyser, etc.) des algorithmes et, par ailleurs, permet de les tester (par l'exécution du programme). C'est une approche prototypique, mais ce n'est pas la seule possible.

2. Pourquoi enseigner l'informatique à l'école primaire ?

J'indique tout d'abord les arguments de différents promoteurs de l'enseignement de l'informatique et ce que l'on trouve dans les programmes actuels, puis propose un point de vue personnel.

2.1. Arguments en faveur d'un enseignement en primaire

Il y a eu ces dernières années différentes demandes de prise en compte de l'enseignement de l'informatique dans les programmes du primaire et du secondaire, co-signées par différents acteurs plus ou moins institutionnels[16]. De façon synthétique :

- Argument clé évoqué pour développer un programme d'informatique à l'école primaire : « Comme dans les autres disciplines fondamentales, la sensibilisation précoce aux grands concepts de la science et technique informatique est essentielle. Elle donne des clés aux élèves pour comprendre le monde qui les entoure, elle évite que se forgent des idées fausses et représentations inadéquates, elle fabrique un socle sur lequel les connaissances futures pourront se construire au Collège et au Lycée ».
- Objectif proposé : la « découverte des concepts fondamentaux de l'informatique », les concepts mis en avant étant ceux de langage, d'information, d'algorithme et de machine.

Il y a eu également un avis de l'Académie des sciences[17] (auteurs proches du précédent, les documents se recoupent largement). De façon synthétique :

- Arguments clés : l'informatique est une discipline scientifique clé ; nécessité de comprendre le monde et « illettrisme informatique » actuel ; retard de la France ; emplois.
- Recommandations / programmes :
  - Primaire : « inclure une initiation aux concepts de l'informatique » (« sensibilisation aux notions d'information et d'algorithme, possible à partir d'exemples très variés dans le style de « La main à la pâte » pour les sciences physiques »).
  - Collège : « introduire un véritable enseignement de l'informatique, qui ne soit pas noyé dans les autres enseignements scientifiques et techniques, mais développe des coopérations avec ceux-ci dans une volonté d'interdisciplinarité ».
  - Approche proposée, 3 phases : « La sensibilisation, principalement au primaire, qui peut se faire de façon complémentaire en utilisant des ordinateurs ou de façon « débranchée » ; un matériau didactique abondant et de qualité est d'ores et déjà disponible. L'acquisition de l'autonomie, qui doit commencer au collège et approfondir la structuration de données et l'algorithmique. Une initiation à la programmation est un point de passage obligé d'activités créatrices, et donc d'autonomie. Le perfectionnement, qui doit se faire principalement au lycée, avec un approfondissement accru des notions de base et des expérimentations les plus variées possibles ».

2.2. Les programmes actuels

Dans le « socle commun de connaissances, de compétences et de culture[18] » on trouve : « [L'élève] sait que des langages informatiques sont utilisés pour programmer des outils numériques et réaliser des traitements automatiques

[16] Cf. le document « Proposition d'orientations générales pour un programme d'informatique à l'école primaire » (2013), http://www.epi.asso.fr/revue/editic/itic-ecole-prog_2013-12.htm

[17] Avis de l'Académie des sciences « L'enseignement de l'informatique en France - Il est urgent de ne plus attendre » (2013) ; www.academie-sciences.fr/pdf/rapport/rads_0513.pdf

[18] « Le socle commun de connaissances, de compétences et de culture couvre la période de la scolarité obligatoire, c'est-à-dire dix années fondamentales de la vie et de la formation des enfants, de six à seize ans. Il correspond pour l'essentiel aux enseignements de l'école élémentaire et du collège qui constituent une culture scolaire commune ».

de données. Il connaît les principes de base de l'algorithmique et de la conception des programmes informatiques. Il les met en œuvre pour créer des applications simples. »[19].

Dans les programmes officiels actuels de cycle 3 on peut trouver les choses suivantes[20] :

- 5 domaines de formation qui définissent les grands enjeux de formation ; domaine 1 = « les langages pour penser et communiquer » (…) « Comprendre, s'exprimer en utilisant les langages mathématiques, scientifiques et informatiques ».
- Plusieurs références à « langage de programmation » en lien avec les maths (e.g., « des activités géométriques peuvent être l'occasion d'amener les élèves à utiliser différents supports de travail : papier et crayon, mais aussi logiciels de géométrie dynamique, d'initiation à la programmation ou logiciels de visualisation de cartes, de plans » ; « Espace et géométrie (…) constituent des moments privilégiés pour une première initiation à la programmation notamment à travers la programmation de déplacements ou de construction de figures ».
- Matériaux et objets techniques / attendus de fin de cycle / « Les élèves apprennent à connaitre l'organisation d'un environnement numérique. Ils décrivent un système technique par ses composants et leurs relations. Les élèves découvrent l'algorithme en utilisant des logiciels d'applications visuelles et ludiques. Ils exploitent les moyens informatiques en pratiquant le travail collaboratif. Les élèves maitrisent le fonctionnement de logiciels usuels et s'approprient leur fonctionnement ».

Les programmes actuels de l'école primaire évoquent donc une initiation à l'informatique. Il n'y a pas de programme précis cependant. On peut également remarquer que c'est assez orienté « maths »[21].

2.3. Point de vue personnel

Mon avis personnel (qui ne vaut pas mieux qu'un autre, mais sous-tend ce document et permet de le mieux le comprendre donc) est que :

- Un certain nombre de notions ou de façons de faire fondamentales et/ou prégnantes en informatique (par exemple, l'algorithmique, la logique, la décomposition de problèmes en sous-problèmes, l'abstraction) sont des compétences utiles et importantes à tous, et pas simplement aux informaticiens.
- Il est possible d'enseigner ces compétences de différentes façons, qui ne sont pas exclusives. L'approche algorithmique/programmation semble présenter des propriétés intéressantes. Elle amène notamment à travailler sur réifications. Par exemple, quand on travaille sur un bout de code et qu'on veut le rendre « plus abstrait », on travaille sur un objet concret, qu'on voit à l'écran et qu'on modifie (cf. les blocs Scratch présentés en Section 1). Il y a un support concret à la réflexion et à l'action.
- Il est donc possible que l'enseignement de l'informatique favorise ces apprentissages plus que d'autres approches et/ou en permette un apprentissage alternatif pertinent pour certains élèves. Mais cela, bien sûr, il faudra l'étudier et le mesurer scientifiquement.
- L'enseignement de l'informatique passe, classiquement, par des exercices de type « écrire un programme qui fait xxx », le fait de « faire xxx » nécessitant que les élèves utilisent les notions algorithmiques que l'on veut leur faire travailler. Ce xxx peut être sans intérêt autre que de créer un cadre à l'exercice ou relever d'autres compétences du programme (en maths, en Français, etc.). Il est donc possible de coupler enseignement de l'informatique et d'autres domaines.
- Par ailleurs, les points d'entrées « informatique » et « programmation » sont des moyens que certains élèves peuvent percevoir (ou percevoir à certains moments) comme ludiques et motivants. Il est possible de faire travailler la notion de boucle en faisant des programmes qui font des calculs réitérés (cf. exemples en section 5). Mais il est également possible que le « xxx » de « écrire un programme qui

[19] Extrait du Bulletin officiel n° 17 du 23 avril 2015.

[20] Synthèse d'éléments extraits du BO Bulletin officiel du 26 novembre 2015, cycle 3, hors compétences = techniques usuelles de l'information et de la communication (spectre du B2i) ; rappel : le cycle 3 se termine maintenant en 6ième).

[21] Comme toujours, les programmes définis à un instant t sont le résultat de considérations et d'arbitrages multiples, en l'occurrence : réflexions sur l'intérêt de l'enseignement de l'informatique en tant que tel ; réflexions sur le niveau où cela doit être enseigné (primaire, collège, etc.) ; réflexions sur la façon dont cela doit être abordé (cf. Section 4.2) ; enjeux humains, disciplinaires et institutionnels (risque de réduction des heures d'enseignement d'autres domaines ; statut de discipline ou de technique de l'informatique ; enseignement par les profs de maths, les profs de techno ou des profs d'informatique ; problèmes de la formation et de la certification ; etc.) ; lobbyings divers.

fait xxx » soit « faire danser le personnage », ce qui va également amener à écrire des boucles (cf. exemple du saut périlleux en Section 1.2).

- En toute hypothèse, les points d'entrées « informatique » et « programmation » permettent d'aborder différents sujets connexes, par exemple : enseigner ce qu'il y a derrière « les ordinateurs » ; démythifier (pas de magie, mais des programmes, faits par des humains, qui appliquent bêtement -ou plutôt, mécaniquement- ce qu'on leur a dit de faire) ; inciter à la réflexion (par exemple, une fois compris ce qu'est un algorithme, qu'un moteur de recherche ne renvoie pas « l'information » sur un sujet mais les données qu'un algorithme particulier a identifiées comme pertinentes et donc, en fait, l'information que quelqu'un a décidé, pour certaines raisons, légitimes ou pas, de renvoyer). Et ça, c'est déjà pas mal.
- L'enseignement de l'informatique peut donc être abordé de différentes façons : en tant que tel ; comme un moyen pour former les élèves à d'autres disciplines et/ou à certaines compétences du socle commun ; et/ou comme un moyen pour atteindre le but général de l'école de former des citoyens éclairés. Ces objectifs ne sont bien évidemment pas exclusifs.

Sur ces bases, à la question « Faut-il enseigner l'informatique à l'école primaire ? », ma réponse est : « Je n'en sais rien ». Je pense que c'est faisable. Je pense que c'est potentiellement pertinent. Mais je ne sais pas si c'est souhaitable, notamment par manque d'études des impacts en termes d'apprentissage (est-ce que les élèves de primaires sont aptes à comprendre les notions impliquées ? est-ce le bon / le meilleur niveau pour les enseigner ? est-ce que l'enseignement « par l'informatique » semble plus efficace pour les élèves, ou pour certains élèves ? est-ce qu'il y a transfert vers d'autres domaines ? etc.) et d'études sur les avantages/inconvénients de faire cet enseignement à ce niveau (plutôt que d'autres enseignements : les programmes sont chargés). Ceci étant, en toute hypothèse, y réfléchir et mener des actions exploratoires en classe est un pas vers le fait de savoir (par le retour des enseignants, par les analyses scientifiques qui pourront être faites).

Il y a actuellement un effet de mode. Il faut donc prendre de la distance par rapport aux discours trop généraux et/ou trop prosélytes (et aux discours caricaturalement négatifs également). Beaucoup visent à faire agir le politique sur des questions de long terme plus qu'à aider les enseignants ici et maintenant. Cependant, prendre de la distance ne veut pas dire jeter le bébé avec l'eau du bain, i.e., refuser de considérer l'enseignement de l'informatique car certains discours sont trop généraux et/ou inexploitables et/ou, pour certains, infondés (on trouve effectivement des discours complètement délirants).

3. Que faut-il faire pour enseigner la pensée informatique à l'école ?

Comme pour tout enseignement, enseigner la pensée informatique nécessite de mener une réflexion pédagogique : il faut disposer ou construire une certaine compréhension et maîtrise du domaine d'enseignement (l'informatique, l'algorithmique, la programmation) et des enjeux d'apprentissage, puis développer une idée claire des objectifs pédagogiques que l'on considère et des moyens de les atteindre. J'aborde ci-dessous ces différents points.

3.1. Développer une certaine compréhension du domaine d'enseignement

Développer une certaine compréhension de la pensée informatique et de son enseignement peut apparaitre comme difficile, mais il ne faut pas s'arrêter aux apparences.

Cela peut apparaitre comme difficile (et donc inquiéter) pour différentes raisons. L'objet à enseigner (la « pensée informatique »), pourquoi et comment on devrait l'enseigner sont encore, si on les compare à d'autres domaines d'enseignement de l'école, mal définis (c'est un domaine récent, il y a débat sur son statut et, surtout, il y a pas/peu d'expérience). De façon générale, et c'est le principal problème, la didactique de l'informatique est très peu développée. Par suite, en tant qu'enseignant, le domaine d'enseignement est perçu depuis et à travers son expérience personnelle. Or, si l'on n'a pas étudié l'informatique en tant que telle dans son parcours d'étudiant et/ou si l'on ne travaille pas la question spécifiquement, la perception par défaut de l'informatique que l'on a est celle de l'utilisateur (d'un ordinateur, de logiciels de bureautique, etc.). Ceci ne permet pas de comprendre l'idée de « pensée informatique » et, pire, crée des confusions. Les choses sont également compliquées par deux facteurs propres à ce domaine. Tout d'abord l'informatique, en tant que domaine scientifique et technique, est un domaine complexe. C'est un domaine dont on peut avoir différentes perceptions (liens avec les maths, liens avec les machines, liens avec les sciences humaines, etc.) ; tous les informaticiens ne sont d'ailleurs pas d'accord entre eux. C'est un domaine qui en recoupe beaucoup d'autres, et c'est parfois difficile de faire la part des choses (par exemple, la notion d'algorithme, ou la logique, relèvent de l'informatique et d'autres domaines ; idem pour les notions de codage et de langage ; etc.). C'est un domaine dont les dimensions scientifiques (notions abstraites, principes, méthodes) et techniques (ordinateur, langage de programmation) se mêlent et, parfois, se confondent. Par ailleurs, l'enseignement de l'informatique est actuellement un objet de réflexion mais aussi un enjeu (scientifique, médiatique, politique). Cela

n'aide pas à développer une vision pédagogique. Par exemple, au niveau secondaire mais cela rejaillit sur le primaire, le débat sur ce qu'il faut enseigner est parasité par les questions de qui va l'enseigner, de légitimité disciplinaire, etc.

Ceci étant, prenons les choses autrement : est-ce que les enseignants de primaire ayant fait une licence/master d'histoire, ou d'anglais, ou de science de l'éducation (…) sont des « mathématiciens » ? Seraient-ils capables de décrire ce que sont « les mathématiques » d'une façon satisfaisant les chercheurs en mathématiques ou, même, les enseignants du secondaire ? Cela ne les empêche pourtant pas d'enseigner la numération ou la géométrie aussi bien que les autres. Inversement, les matheux sauraient-ils conceptualiser tous les différents genres littéraires ? (etc.).

Pour enseigner la pensée informatique en primaire, il n'est pas nécessaire de tout savoir et tout comprendre de l'informatique, ni des débats actuels sur son statut. Les professeurs des écoles n'ont pas à devenir plus informaticiens qu'ils ne sont matheux, historiens, ou spécialistes des différentes matières enseignées à l'école. Il faut et il suffit de :

1. Comprendre le type de compétences sous-jacentes à la « pensée informatique » que l'on se propose de faire travailler aux élèves.
2. Etre capable de conceptualiser (identifier, définir, dissocier, etc.) ce que l'on veut enseigner, i.e., de raisonner en terme d'objectifs pédagogiques.
3. Etre capable d'identifier les moyens utiles pour atteindre ces objectifs pédagogiques (types d'exercices, etc.) ce qui inclut, éventuellement, de savoir utiliser le *moyen* « langage de programmation » (comme on utilise le moyen « calculatrice » dans certains exercices de maths, ou le moyen « dictionnaire » dans certains exercices de Français).

Sur ces différents points, le fait de ne pas avoir suivi de cours d'informatique dans son cursus universitaire ne pose pas de problème insurmontable[22]. Ce peut même être un avantage car, dans certains cursus, les étudiants apprennent à programmer mais pas à comprendre les concepts abstraits sous-jacents, et développent des conceptualisations de l'informatique très éloignées de la « pensée informatique », qui n'aident pas à son enseignement.

3.2. Raisonner en termes d'objectifs pédagogiques

Raisonner en termes d'objectifs pédagogiques est, à mon sens, une difficulté plus importante que celle de maitriser le domaine d'enseignement au niveau requis.

La difficulté de surface est l'absence de programme clair pour le cycle 3. Ceci étant, les notions qui forment les « blocs » à partir desquels on construit des algorithmes sont peu nombreuses, et on va pour l'essentiel retrouver les mêmes à l'école, collège, lycée et Université.

Si l'on prend l'axe des programmes officiels, il est donc possible de s'inspirer du programme du cycle 4 et d'aborder en CM1/CM2, voire en CE, les notions suivantes (déjà présentées dans leur version Scratch en Section 1)[23] :

- Notion d'action / instruction (« dire », « avancer », « jouer un son », etc. ; à choisir en fonction du type d'exercice abordé : faire des calculs, faire se déplacer un personnage, etc. Certaines actions utilisent des variables prédéfinies : les actions de déplacement utilisent des variables x et y correspondant au positionnement du personnage, l'action « demander » utilise la variable « réponse ».
- Structure conditionnelle (Si-Alors-Sinon) et boucle (Répéter[24]), et les notions associées (expressions logiques, opérateurs numériques et logiques). Il est possible de n'aborder que des choses simples (« Si la valeur de l'âge est inférieure à 18 Alors … », « Répéter 10 fois ») ou plus complexe (« Si la valeur de l'âge est supérieure à 10 et inférieure à 18 », « Répéter jusqu'à … »).

[22] Je parle ici de l'enseignement de l'algorithmique/programmation à l'école tel qu'abordé dans ce document. L'enseignement de l'informatique en tant que discipline, dans ses différentes dimensions, c'est autre chose bien sûr.

[23] Pour ce qui est la construction d'algorithmes, la différence entre les niveaux d'enseignement n'est pas tant en termes de notions algorithmiques que de difficulté des problèmes abordés et des algorithmes construits. Le choix des notions à aborder, la progression et la difficulté des exercices sont à évaluer en fonction des capacités des élèves, du nombre de séances qui y sont consacrées et des enseignements des années précédentes (enseignement de l'algorithmique et/ou familiarisation avec la manipulation du clavier et de la souris et/ou familiarisation avec l'environnement Scratch). Bien évidemment, il y a beaucoup d'autres choses en informatique qui ne sont absolument pas abordable en primaire, dont d'autres aspects de l'algorithmique (correction, complétude, preuve, test, complexité, etc.).

[24] Scratch propose 3 variantes de la notion de boucle (appelée aussi « structure itérative ») : « Répéter 10 fois», « Répéter jusqu'à ce que telle condition soit vérifiée » et « Répéter indéfiniment ».

- Mécanismes de lancement de base : lancer le script quand on appuie sur le « drapeau vert » ou sur le personnage concerné (en termes Scratch).
- Ensuite, selon le temps consacré : utilisation de variables[25] (variable « réponse » ou définies par l'élève ou l'enseignant, cf. exemples illustratifs en Section 5) ; mécanismes de synchronisation : via des délais (attendre x secondes puis faire telle action) et/ou via des envois de messages entre personnages (cf. exemples illustratifs en Section 5 et, pour les détails, la documentation Scratch)[26].

Cependant, identifier les notions objets d'enseignement ne clarifie que partiellement les choses. En effet, il est possible d'aborder l'enseignement de ces notions de façons très différentes. Par ailleurs, et surtout en primaire, ce serait sans doute dommage de ne voir l'enseignement de l'informatique que sous l'angle de l'enseignement de l'algorithmique. En effet, et c'est sans doute l'une des particularités de l'informatique, il est extrêmement facile de définir des exercices mêlant informatique (algorithmique, etc.) et d'autres domaines (maths, Français, langues, arts, etc.) ou dimensions (compétences transversales comme la capacité d'analyse, la résolution de problèmes, l'écriture, la méthodologie, l'organisation, le travail collaboratif, la créativité, etc.).

La difficulté à raisonner en termes d'objectifs pédagogiques va alors tenir à la richesse des situations, la pluralité des objectifs que l'on peut considérer et/ou à leur intrication potentielle. Ce point « objectifs pédagogiques » est développé en Section 4.

L'identification des objectifs pédagogiques que l'on considère permet, ensuite, de tirer profit des manuels existants (il en existe, je n'ai donc pas cherché à en faire un de plus) et/ou des idées que l'on peut tirer de la multitude d'exemples développés au sein de la communauté de pratique autour de Scratch. Ce point est développé en Section 7.

3.3. Savoir utiliser un langage de programmation

Si l'on reste sur de l'informatique sans ordinateur, il n'y a pas d'aspect technique particulier.

Si l'on veut faire développer des programmes aux élèves, il faut savoir utiliser un langage de programmation et, surtout, savoir comment l'utiliser comme un moyen pour atteindre les objectifs pédagogiques considérés.

Ici encore, les aspects techniques peuvent inquiéter, mais il ne faut pas s'arrêter aux apparences (savoir utiliser un langage comme Scratch n'a rien à voir avec la difficulté de comprendre les langages classiques).

Par ailleurs et surtout, il faut bien comprendre le point suivant. Il est possible (et, personnellement, cela me semble une bonne idée) de ne pas se donner comme objectif d'enseigner le langage de programmation en tant que tel, mais de se donner comme objectif de savoir aider/guider les élèves à utiliser un *moyen*. Le langage de programmation peut donc être abordé et traité comme une ressource, que l'enseignant peut plus ou moins maitriser (ce qui n'est pas le cas pour un objet d'enseignement bien sûr). Ce point est développé en Section 6.

Au niveau du primaire, le langage/environnement de programmation habituellement utilisé est Scratch. Il en existe d'autres. Il se trouve que celui-ci présente un certain nombre de propriétés intéressantes :

- Scratch propose les structures algorithmiques de base et des mécanismes de synchronisation clairs, propres et suffisants. Contrairement à ce qu'on lit parfois, Scratch est un vrai langage de programmation (en terme de paradigmes : programmation impérative et événementielle).
- Scratch a été développé pour les enfants : il a une dimension ludique, il est pensé pour être facile à utiliser, il élimine les problèmes de saisies inutiles et de syntaxe.
- Il y a une immense communauté internationale de gens (enseignants, formateurs, animateurs, élèves, enfants) qui l'utilisent, ce qui permet de récupérer et/ou partager de la documentation, des exemples, des idées, des programmes[27]. Le fait qu'il y ait une communauté de pratique est un atout majeur.
- Il a, en tant qu'environnement technologique, une certaine stabilité, et va probablement être accessible et utilisable longtemps. Travailler avec un langage/environnement informatique ésotérique (« nouveau

[25] Une institutrice me signale que la notion de variable est pour beaucoup d'élèves difficile à comprendre. C'est typiquement le type de notion qui pour certains élèves est immédiate et pour d'autres bloque complètement.

[26] Les repères de progressivité du cycle 4 indiquent, en creux, les éléments à ne pas dépasser en cycle 3 (e.g., pour la 5e, « traitement, mise au point et exécution de programme simple avec un nombre limité de variables d'entrée et de sortie, développement de programmes avec des boucles itératives »).

[27] Ceci permet, si c'est l'une des modalités de fonctionnement de la classe/école, de travailler sur des projets ou d'échanger des productions avec d'autres classes/écoles, en France ou à l'étranger (en travaillant au passage les langues étrangères). C'est particulièrement facile avec Scratch qui permet de créer et partager des zones de dépôt/réutilisation de programmes.

et absolument génial, tout juste inventé par 2 passionnés »), c'est prendre le risque que dans quelque mois cela ne marche plus (les passions, cela passe ; derrière Scratch, il y a des institutions solides).

- Les gens qui l'ont développé (Media Lab du MIT) sont des gens qui sont forts, qui ont beaucoup réfléchi et beaucoup travaillé. Scratch, comme tout langage/environnement, est le résultat de compromis dont les justifications échappent souvent aux utilisateurs … mais sont bien fondées (Cf. Section 4.2).
- Utiliser Scratch en ligne ou l'installer sur les ordinateurs de l'école (si l'on n'utilise pas la version en ligne) est trivial. D'un point de vue technique, utiliser l'outil Scratch est probablement plus facile que d'utiliser l'outil TBI (la difficulté pédagogique à exploiter ces outils, c'est une autre question bien sûr).

Ceci étant, rien n'empêche d'utiliser un autre langage si l'on a des bonnes raisons pour cela.

3.4. Synthèse

Pour ce qui est des connaissances disciplinaires nécessaires à l'enseignement de l'informatique, celles-ci ne sont pas plus compliquées que de celles d'autres matières (maths, français, histoire, etc.) au même niveau d'enseignement. La situation est simplement différente du fait qu'il s'agit, pour beaucoup d'enseignants qui ne les ont pas rencontrées dans leur parcours d'écolier et d'étudiant, de les découvrir et de les assimiler dans le cadre de leur formation d'enseignant. Mais bon, on parle ici d'enseigner des notions et des façons de faire du niveau du primaire, cf. Section 1[28].

Pour ce qui est des aspects d'enseignement, le travail qui consiste à aider les enseignants (études didactiques, réunions de consensus, retours d'expériences, etc.) est actuellement (très) peu avancé, du moins en France, et une large part reste donc à la charge de l'enseignant. Vu l'effet de mode actuel, on peut cependant penser qu'il va y avoir très rapidement pléthore de manuels. Il y a cependant déjà des ressources « clés en main », cf. Section 7.

L'informatique est à la fois un objet d'enseignement (comprendre la notion de boucle) et un moyen de faire des choses en maths, en Français, en arts, etc. Cela ouvre un large champ pour l'exploration d'innovations pédagogiques tissant des liens entre différents apprentissages (exercices ou de projets multi-objectifs et pluridisciplinaires)[29]. A ce niveau, le contexte du primaire, où un même enseignant organise et conduit un enseignement polyvalent, semble particulièrement propice.

A mon sens, (1) dépasser l'éventuelle mauvaise conceptualisation de l'informatique développée en tant qu'utilisateur et (2) raisonner en termes d'objectifs pédagogiques sont les deux difficultés principales. Le second point lève par ailleurs la difficulté supplémentaire de trouver la place de l'informatique, en tant que telle et/ou en lien avec d'autres matières, au sein de l'ensemble des enseignements.

La section suivante essaie d'apporter une aide sur ces différents points.

4. Quels objectifs pédagogiques peut-on considérer ?

Il est possible de construire des séquences pédagogiques abordant de l'informatique avec des objectifs pédagogiques très différents. Dans cette section, je commence par dresser une liste d'objectifs possibles (la différentiation d'objectifs possibles aidant à conceptualiser le domaine je pense). Ensuite, j'essaie de prendre du recul et de mettre en évidence des différences d'approches de l'enseignement de la pensée informatique.

4.1. Différents objectifs possibles

Le tableau ci-dessous liste quelques possibilités d'objectifs pédagogiques, à prendre comme support de réflexion : ces objectifs ne sont ni exclusifs ni disjoints ; certains objectifs sont (aussi) des moyens pour atteindre d'autres objectifs ; il serait possible de présenter certains points autrement. Attention : cette liste d'objectifs n'est pas focalisée sur le cycle 3.

[28] Les choses sont différentes pour les niveaux collège/lycée, où il est possible de travailler sur des algorithmes plus complexes mais, surtout, où d'autres aspects (techniques notamment) doivent être abordés.

[29] Différents acteurs peuvent et doivent jouer un rôle dans la réflexion et les travaux sur l'enseignement de l'informatique au primaire (chercheurs, formateurs de formateurs, politiques) ; le rôle des enseignants est fondamental.

	Objectif	Remarque
1	faire comprendre la notion d'algorithme et les notions sous-jacentes (action/instruction, séquence, boucle, structure conditionnelle, etc.)	ce type d'objectif peut être abordé en regardant et analysant des algorithmes et/ou en construisant des algorithmes (deux compétences différentes) ; il est possible de travailler sur des algorithmes « de tous les jours » (recette de cuisine, multiplication d'entiers, accord du participe passé) et/ou des choses plus spécifiques (tri, recherche dichotomique, etc.)
2	faire pratiquer la construction d'algorithmes	cf. remarque ci-dessus : inventer/construire un algorithme et comprendre un algorithme sont deux choses très différentes
3	faire comprendre la notion de langage (langage naturel, langage informatique) en tant que telle (comme un système de codage, comme un outil de pensée) et donc l'existence et l'utilité d'une multiplicité de langages	ce type d'objectif (que l'on trouve dans certains documents) semble ambitieux, mais peut être abordé sur le mode d'une première sensibilisation, comme un à-côté (un effet de bord) de la pratique de la programmation (?)
4	faire comprendre la notion de langage de programmation, pratiquer la programmation	ce type d'objectif peut être abordé avec un langage de programmation exécutable sur ordinateur comme Scratch ou un langage structuré type « langage de commande de robot » (avancer, tourner-Droite, tourner-Gauche, etc.)
5	faire en sorte que les élèves sachent programmer dans un langage de programmation (par exemple, Scratch)	c'est un objectif qui peut/doit être couplé avec ceux liés à l'algorithmique (et/ou la créativité) ; attention, le considérer en tant que tel uniquement est très réducteur (savoir utiliser Scratch sans faire le lien avec des principes un peu abstraits ou, en termes informatiques, « bidouiller », n'a que peu d'intérêt et peut amener à développer une pauvre compréhension du domaine)
6	faire comprendre et pratiquer la démarche idée générale / modélisation / algorithme / codage (et, éventuellement, décomposition en sous-problèmes/équipes)	ce type d'objectif peut être abordé comme un cas particulier de « penser puis agir » ; il peut être mis en œuvre sur une séance ou (plus probablement) sous la forme de projets sur plusieurs séances
7	faire comprendre des objets et/ou logiciels d'usage courant comme une page Web, une messagerie électronique, un moteur de recherche, un appareil photo numérique, etc. et/ou faire comprendre comment fonctionne un ordinateur et un réseau d'ordinateurs	ce type d'objectif peut être abordé de différentes façons : comme un moyen de démythifier, de sensibiliser à des questions d'éthique ; comme une connaissance nécessaire à l'honnête citoyen ; et/ou comme un support pour enseigner des notions informatiques (information, algorithme, ordinateur = exécuteur d'algorithmes / de programmes, tryptique données – traitements - résultats, représentation binaire, stockage d'information, processus de calcul, aspects matériels, etc.)
8	sensibiliser à des questions d'éthique, de principes, etc.	par exemple, dans le contexte de l'analyse ou de la création de programmes, faire réfléchir les élèves sur les constructions collectives/privées, la notion de logiciel libre/propriétaire (Scratch pousse à partager ses programmes), la qualité (véracité, précision, etc.) de ce que calcule un programme, etc.
9	faire travailler une compétence (informatique ou pas) en l'introduisant comme un moyen nécessaire pour l'écriture d'un algorithme / d'un programme	par exemple, faire travailler une compétence en logique (conjonction, disjonction, négation), en maths, (etc.) en travaillant sur un algorithme où l'écriture d'une expression logique (ou le calcul de maths, ou etc.) est nécessaire, et en utilisant le fait que Scratch propose des moyens supports à l'écriture d'expressions logiques (de calculs, etc.)
10	faire travailler une compétence (informatique ou pas) en utilisant la programmation comme un contexte ludique	par exemple, faire travailler la production de texte en créant des scènes où différents personnages discutent
11	faire en sorte que les élèves développent leur créativité	c'est un objectif phare dans la vision US et, du coup, dans beaucoup de manuels/ressources

12	faire en sorte que les élèves sachent exprimer leurs idées « créatives » (jeu, etc.) en termes de « problème à résoudre » et de moyens systématique de résoudre ces problèmes (i.e., d'algorithme et de programme)	cela rejoint l'objectif « faire pratiquer la construction d'algorithmes », mais c'est une modalité différente ; c'est assez différent de « savoir écrire un algorithme qui fait xxx », xxx étant spécifié par l'enseignant, et de « savoir exprimer sa pensée sous forme d'algorithme » ; pour prendre une métaphore, c'est différent de faire une dictée où le texte est imposé et d'imaginer et d'écrire un texte où je choisis mes phrases (ce qui me permet au passage d'éviter celles où je ne sais pas accorder le verbe) ; mais les choses se rejoignent à un certain niveau bien sûr
	contre-exemple : faire comprendre et pratiquer l'utilisation d'un ordinateur et des logiciels de bureautique	c'est l'objet du B2i, et clairement pas l'objet abordé dans ce document
	contre-exemple : faire utiliser un programme Scratch à des élèves	l'utilisation d'un programme tout fait n'amène en général pas à travailler des compétences informatiques ; cf l'exemple des fractions ci-dessous
	contre-exemple : faire en sorte que les élèves deviennent « informaticiens »	c'est quoi un « informaticien » ? il ne faut pas confondre les compétences générales de l'honnête citoyen » (lire, écrire, compter, résoudre des problèmes, comprendre x et y, etc.) et activité professionnelle ou hobby (enseigner l'écriture à l'école primaire, ce n'est pas former des écrivains, même si certains élèves deviendront peut être écrivains)
	contre-exemple : faire programmer aux élèves le jeu xxx (on trouve beaucoup de cas où xxx= « Pong[30] », il semble même que ce soit un exemple proposé dans les programmes)	présenté comme cela, c'est sans intérêt bien évidemment ; à moins que l'objectif affiché (programmer Pong) ne cache une réflexion ayant permis d'identifier des objectifs pédagogiques précis, et que l'exercice et le contexte soient pensés et étudiés comme des *moyens* pour que les élèves travaillent les compétences cibles ; on revient alors vers un ou plusieurs des objectifs précédents

4.2. Différentes conceptions/approches de l'enseignement de la « pensée informatique »

La liste présentée ci-dessus illustre qu'il est possible de considérer des objectifs de natures très différentes. Sans entrer dans des considérations épistémologiques trop complexes il est possible de distinguer et d'esquisser deux approches et/ou conceptions, toutes les deux légitimes et qui se recoupent, mais sont cependant assez différentes[31].

Approche orientée algorithmique

Il y a une approche (et/ou conception) de la notion de « pensée informatique » qui est très proche de la discipline informatique telle qu'elle existe (dans l'industrie, dans la recherche), est enseignée à l'Université et, pour certains (cf. supra), devrait être enseignée aux primaire et secondaire. Dans cette approche, et en se focalisant sur le primaire, l'idée centrale est que le cœur de ce qu'il faut enseigner, c'est l'algorithmique et les notions/compétences sous-jacentes (identification/modélisation des données ; abstraction/généralisation ; notions de variables, boucles, conditionnelle ; etc.).

Cette première conception peut être plus ou moins liée à l'ancrage technologique que constituent les ordinateurs. Dans une vision/mise en œuvre techno-centrée, l'objectif *in fine* est que les élèves sachent comprendre/faire des programmes informatiques, ce qui nécessite de (1) comprendre/savoir faire des algorithmes et (2) comprendre/savoir utiliser un langage de programmation. Les activités pédagogiques sans ordinateurs sont alors des moyens pour enseigner des notions et principes qui seront ensuite réinvestis ; les exercices proposés aux élèves sont typiquement conçus pour les amener à mobiliser telle ou telle compétence algorithmique cible ; et les langages de programmation sont des moyens d'écrire des algorithmes sous une forme exécutable par un ordinateur. Dans une

[30] Jeux vidéo où l'on renvoie une balle contre un mur à l'aide d'une raquette. Il semble que pour les élèves d'aujourd'hui les exemples soient plutôt à chercher du côté des « Angry Birds » (?).

[31] La complexité vient, entre autre, de la multiplicité des facteurs : conceptuels (conception de ce qu'est l'informatique), institutionnels, politiques, culturels.

vision/mise en œuvre plus centrée sur l'apprentissage de mécanismes d'analyse et de résolution de problème, les choses peuvent s'inverser : les activités sans ordinateur peuvent être utilisées sans passage sur machine, et la réalisation d'un algorithme programmé sur machine peut être vue comme un moyen de développer des compétences plus générales, non limitées au domaine informatique. Ces deux visions ne sont pas nécessairement en opposition et peuvent cohabiter.

On peut noter que beaucoup des arguments en faveur de l'enseignement de l'informatique, propositions de programmes (etc.) que l'on peut lire en France sont plutôt ou essentiellement sous-tendus par cette approche. On trouve également des choses similaires dans des textes produits par des associations pour l'enseignement de l'informatique aux USA[32].

Approche orientée créativité

Il y a une autre approche (et/ou conception) de la notion de pensée informatique (et de l'usage des langages de programmation) qui prend un point d'entrée différent, plus général. Une façon de comprendre cette conception est la vision (US) des 4 compétences générales clés (« 21st century skills ») que sont « the four C's" : "Critical thinking and problem solving", "Communication", "Collaboration", and "Creativity and innovation". En prenant ce point d'entrée, pensée informatique et algorithmique peuvent être abordés et enseignés comme des compétences relevant, en tant que telles et comme moyens, de ces « 21st Century challenges ».

L'idée clé est alors de placer l'élève en situation de créativité et de proposer l'informatique comme un moyen pour exprimer cette créativité. L'objectif est que les élèves développent un lien personnel avec l'informatique, que ce soit pour eux une façon de s'exprimer (et, par suite, de penser, de comprendre, en faisant appel aux notions ou façons de faire de l'informatique). Cette seconde conception met l'accent sur d'autres dimensions donc. Elle est maintenant généralement référencée comme « l'informatique créative » (traduction de « Creative Computing », qui est notamment le titre d'un livre-référence sur l'utilisation de Scratch)[33].

Les promoteurs de cette approche, sur la base d'études des projets construits par les élèves/enfants (en contexte institutionnel ou pas), argumentent que ce type d'activité les amène à pratiquer/aborder des « Computational Concepts: sequences, loops, parallelism, events, conditionals, operators, data", des "Computational Practices: being incremental/iterative, testing/debugging, reusing/remixing, abstracting/modularizing" et des « Computational Perspectives: expressing, connecting, questioning »[34].

Deux points encore pour faire comprendre l'idée. Les auteurs ne parlent pas de faire faire des algorithmes ou des programmes aux élèves, mais de produire des « medias interactifs ». Et, sur une enquête où ils ont demandé à des enfants utilisant Scratch en dehors de l'école avec quelle matière étudiée en classe ils font le rapprochement, la réponse est : arts (n = 20), lecture (n =10), maths (n = 8), science (n =5) (…) informatique (n =2).

On peut noter le lancement en 2015 d'un « enseignement d'exploration d'informatique et de création numérique destiné aux élèves de seconde générale et technologique » (mais, au vu du texte du CSP[35], la notion de « créativité » et l'esprit général semblent assez différents).

Discussion

Avoir en tête les deux conceptions (orientation algorithmique et créativité) esquissées très sommairement ci-dessus est utile pour trois raisons au moins. D'une part, les discours généraux qui mélangent sans s'en rendre compte les idées sous-jacentes à ces deux approches sont souvent porteurs de confusion. Par ailleurs, et surtout, cela amène à réfléchir sur la façon dont on va aborder l'enseignement. Enfin, cela permet de comprendre et donc de mieux exploiter les ressources pédagogiques existantes (soit directement, soit en les détournant ; cf. Section 7).

Il est bien évident que ces deux approches ne sont pas contradictoires. Elles sont cependant différentes. Si l'on se place dans le cadre de la première approche, le point d'entrée standard pour enseigner une compétence cible (par exemple, la notion algorithmique de boucle) et, comme en maths ou d'autres domaines, de faire travailler les élèves sur un exercice identifié comme un vecteur d'apprentissage de cette notion. Cependant, bien entendu, il est possible

[32] Cf csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf

[33] L'informatique créative (2011 puis 2013 ?), traduction Française du livre « Creative Computing Curriculum Guide » ; http://scratched.gse.harvard.edu/resources/informatique-créative

[34] K. Brennan and M. Resnick. New frameworks for studying and assessing the development of computational thinking. In Proceedings of the AERA conference, 2012.

[35] http://www.education.gouv.fr/cid89179/projet-de-programme-pour-un-enseignement-d-exploration-d-informatique-et-de-creation-numerique.html

de développer cette stratégie en abordant des projets liés à des domaines sensés intéresser les élèves (par exemple, le fameux Pong). Le point d'entrée naturel de l'informatique créative est d'amener et d'aider les élèves à développer un projet personnel. Mais, bien entendu, rien n'empêche de conduire l'enseignement pour faire en sorte que les élèves mobilisent des compétences/notions cibles. On arrive donc très vite, quelle que soit l'approche, au fait de travailler sur des projets ayant une certaine durée (et sur ce point il faut se rapporter à la littérature sur la pédagogie par projet, on n'a pas attendu l'informatique pour réfléchir à tout cela !). Ceci étant, on voit facilement que la façon d'aborder le domaine selon l'une ou l'autre approche/conception va amener à considérer différents critères, modalités et/ou priorités.

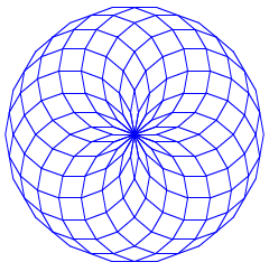
Pousser l'analyse plus loin nécessiterait de développer une réflexion épistémologique qui sort du cadre de ce document[36]. Je liste donc simplement quelques éléments de réflexion complémentaires.

On voit facilement que ces approches pourraient être analysées en termes d'apprentissage implicite/explicite, et qu'il serait également possible et intéressant d'étudier comment coupler apprentissage institutionnel (explicite ?) en classe et activités parascolaires (je ne développe pas ce point plus avant car cela ouvre tout un champ de discussions et de problèmes pédagogiques, institutionnels et organisationnels).

On peut noter que les principales ressources pédagogiques actuellement disponibles, qui sont des traductions de manuels US, relèvent clairement de l'approche « informative créative ». Beaucoup sont exploitables quelle que soit l'approche, mais peuvent donc nécessiter des adaptations.

On peut également noter que Scratch, qui fait la quasi-unanimité comme langage de programmation à faire utiliser aux enfants, y compris auprès de beaucoup des tenants de l'approche « algorithmique » … n'est pas qu'un langage de programmation « facile à utiliser ». C'est effectivement un langage de programmation et, à ce titre, il peut être comparé à d'autres langages à vocation pédagogique ou pas (AlgoBox, Java, Python, etc.). Mais il n'a pas été conçu que pour cela. Il a été conçu comme un outil au service de la vision constructioniste de l'apprentissage développée par Seymour Papert (et, donc, par filiation, la vision constructiviste de Jean Piaget)[37]. Du point de vue des théories de l'apprentissage et de l'enseignement, le prédécesseur de Scratch, c'est la tortue Logo. Beaucoup de ses propriétés, qui apparaissent inutiles voire inappropriées du point de vue du codage d'algorithmes (par exemple les notions de scènes, les « costumes » des personnages, etc.), sont liées à cet objectif. Scratch est conçu pour permettre à des élèves d'exprimer leur créativité par la programmation, de travailler sur des projets qui font sens pour eux et de développer les compétences (résolution de problème, etc.) associées à ce type d'activité, et pas simplement pour programmer des algorithmes (même si, par ailleurs, il permet de programmer des algorithmes).

Il est enfin possible de développer une approche consistant à faire travailler les élèves sur des compétences précises, mais de concevoir des exercices ayant une dimension ludique. Par exemple, des exercices dont le but tel que présenté à l'élève est de faire un programme qui va faire se déplacer ou danser un personnage (actions « avancer », « tourner », « aller à telles coordonnées », « jouer un son »), de tracer des lignes (actions « mettre le stylo en écriture », etc.) … mais qui nécessitent pour cela d'utiliser des boucles, des structures conditionnelles ou des variables. On retrouve ici l'esprit de Logo, mais avec des moyens techniques qui n'ont plus rien à voir (pas plus puissants mais plus faciles, plus jolis, plus multimédia, plus « fun »). Ci-dessous, un exemple classique « à la Logo », dessiner une rosace. Mais ce serait sans doute dommage de ne pas exploiter les actions multimédia de Scratch !

	repete 18 [td 20 repete 18 [td 20 av 20]]	
La rosace dessinée[38].	Le programme en Logo.	Le programme en Scratch.

[36] Analyse qui est, là aussi, rendue délicate par les enjeux disciplinaires, institutionnels, etc. Par exemple, faut-il défendre l'enseignement de l'informatique en mettant en avant l'informatique créative, au risque de la décrédibiliser aux yeux de certains ?

[37] De façon très synthétique, le constructionisme est une théorie de l'apprentissage qui reprend les principes constructivistes et développe l'idée que l'engagement dans des projets où les élèves construisent (et partagent) des objets faisant sens pour eux est un facteur particulièrement favorable à l'apprentissage.

[38] Cet exemple illustre la possibilité d'utiliser Scratch pour la compétence « construction de figures » évoquée dans les programmes

4.3. Quelques éléments supplémentaires liés à mon point de vue personnel

J'aime bien l'idée d'utiliser l'informatique (algorithmique et programmation) pour faire travailler la logique en tant que compétence transverse générale. En informatique, dès que l'on veut faire des conditionnelles (« dans tel cas il faut faire ceci et sinon il faut faire cela ») ou des boucles (« il faut répéter ces actions tant que telle condition est vérifiée »), on manipule des expressions logiques : des conjonctions (des « et »), des disjonctions (des « ou ») et des négations (des « non »). Dans la vie de tous les jours, cela peut aider à éviter des erreurs de raisonnement et/ou à faire ou accepter des argumentations foireuses (affirmation à caractère général sur la base d'une négation de conjonctions transformée en conjonction de négations, etc.).

J'aime bien l'idée de faire réfléchir/travailler les élèves sur les notions d'abstraction (processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise), de modélisation (activité de définition des abstractions pertinentes) et de modèle (abstraction de la réalité, vue subjective de la réalité qui reflète des aspects importants de la réalité). La modélisation, c'est fondamental en informatique, en sciences et, aussi, dans la vie de tous les jours. En particulier, j'aime bien l'idée de faire réfléchir/travailler sur l'idée qu'il n'y a pas de bons et de mauvais modèles, mais des modèles plus ou moins adéquats en fonction de ce que l'on veut faire ; qu'utiliser un modèle inadéquat cela amène à des solutions erronées ; et, aussi, qu'imposer son modèle (son point de vue, sa façon de voir le monde) cela peut être inapproprié, voire dangereux.

J'aime bien l'idée de travailler sur des « projets » = passer plusieurs séances à comprendre/construire ; avec une approche incrémentale et « agile » (faire pour mieux comprendre ce qu'il faut faire, et le faire mieux) ; en se répartissant les tâches et en programmant à deux (là, c'est aussi le chercheur en apprentissage collaboratif qui parle sans doute, mais on trouve également ce type de conseil dans les documents relatifs à Scratch) ; en maintenant un « cahier de conception » des idées, des modèles, des choix que l'on a faits, des échecs, etc. (ce qui permet, au passage, d'utiliser des outils de bureautique et de pratiquer la production de textes).

En passant, une technique de gestion de projet classique en informatique et qui peut facilement être mise en classe. Identifier la liste des tâches à réaliser et les écrire sur des post-it (éventuellement, définir des priorités et/ou des degrés de difficulté, indiquer la nature de la tâche, etc.). Utiliser un grand tableau (une grand feuille de papier) de trois colonnes pour identifier les tâches « à faire » (tous les post-it sont là au début), « en cours » et « terminées». Attribuer les tâches à des élèves ou des binômes et/ou laisser les élèves décider des tâches qu'ils-elles veulent réaliser (en ajoutant le nom de l'élève sur le post-it par exemple). Faire des réunions collectives (par exemple au début et à la fin de chaque séance) pour distribuer les tâches, analyser où l'on en est, faire passer les post-it d'une colonne à une autre, définir de nouvelles tâches, etc. Outre le fait de proposer un support au travail collaboratif, l'intérêt du tableau est de réifier la notion immatérielle de « projet » et son avancement, et de servir du support à la réflexion[39].

5. Que veut dire « utiliser Scratch pour enseigner la pensée informatique » ?

L'objectif de cette section est essentiellement d'étudier ce que peut vouloir dire « faire un exercice d'informatique et utiliser Scratch comme support ». Je me place donc plutôt dans le cadre de la première des approches évoquées ci-dessus. Après des définitions permettant de dresser un cadre commun, je présente différents exemples[40] qui, en plus d'être illustratifs, abordent des points particuliers (pertinents quelle que soit l'approche générale adoptée je pense).

5.1. Définitions et principes généraux

Dans cette section, je vais m'appuyer sur les définitions suivantes[41] :

- Une situation pédagogique est une situation conçue pour amener un apprenant à développer une activité favorable à l'atteinte d'un ou plusieurs objectifs pédagogiques.
- Concevoir et gérer une situation pédagogique consiste généralement à :

(il faut évidemment commencer par des figures plus simples, cf. les ressources pédagogiques Logo).

[39] Pour creuser, chercher des informations sur la « méthode Scrum » et des images de « sprint backlog ».

[40] Les exemples illustrent par ailleurs les notions d'un « programme possible » évoquées en Section 3.2, y compris celles-qui relèvent d'une utilisation avancée (variables, envois de messages). Ils sont accessibles en ligne sur : https://scratch.mit.edu/studios/1808925/.

[41] Reprises et simplifiées de : « Précis de recherche en ingénierie des EIAH » (P. Tchounikine, 2009 ; en ligne à http://lig-membres.imag.fr/tchounikine/).

- Fixer le ou les objectifs pédagogiques considérés.
- Définir la tâche proposée aux élèves (la « tâche-élève »), i.e., ce que les élèves doivent faire. Cette tâche et son contexte de définition doivent être étudiés pour que la réalisation de la tâche amène les élèves à développer une activité (cognitive, comportementale) propice à l'atteinte des objectifs pédagogiques fixés. La tâche-élève peut être proposée à un élève ou à un groupe d'élèves, et elle peut se décomposer en sous-tâches (je resterai sur le singulier pour simplifier le texte).
- Conduire l'enseignement, i.e., faire en sorte que l'activité développée par les apprenants permette l'atteinte des objectifs fixés (ce qui consiste en général à faire en sorte que l'activité des élèves corresponde à l'activité espérée, mais pas toujours). C'est la « tâche-d'enseignement ».

Scratch est un langage/environnement de programmation : c'est un *moyen* pour réaliser des tâches.

De façon générale, utiliser Scratch pour l'enseignement consiste à proposer aux élèves une tâche à réaliser, Scratch étant proposé ou imposé comme un moyen permettant de réaliser tout ou partie de la tâche d'une façon cohérente avec l'activité espérée des élèves, i.e., l'activité que l'on a identifiée comme étant propice à l'atteinte des objectifs pédagogiques fixés.

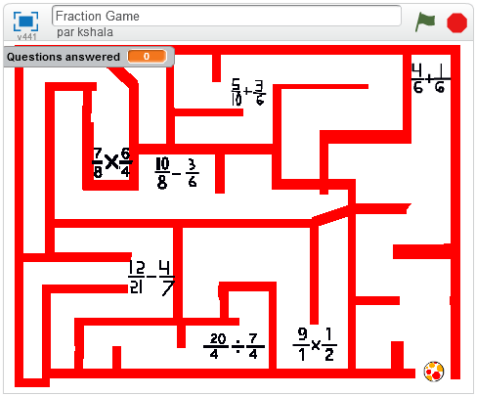
De façon plus précise, utiliser Scratch pour enseigner la pensée informatique c'est proposer une tâche qui amène les élèves à raisonner en termes de problèmes/sous-problèmes (…) et, à un certain moment, en termes d'algorithmes et de programmes, sur papier puis avec Scratch, ou directement avec Scratch.

5.2. Un exemple pour comprendre la différence entre utiliser et faire un algorithme/programme

Pour bien comprendre la différence entre une tâche-élève de type « faire utiliser un programme » et une tâche-élève de type « construire un programme », voici 3 exemples d'utilisation de Scratch ayant pour objectif commun de faire travailler les élèves sur la notion de fraction.

Un jeu (pris sur le site de la communauté Scratch)

Le principe du jeu est faire avancer le ballon (ici en bas à droite) dans le labyrinthe, à l'aide des flèches du clavier. Pour sortir du labyrinthe il va falloir ouvrir des portes en donnant (quand on arrive dans cette zone) le résultat de calculs (par exemple, en haut à droite, 4/6 + 1/6).



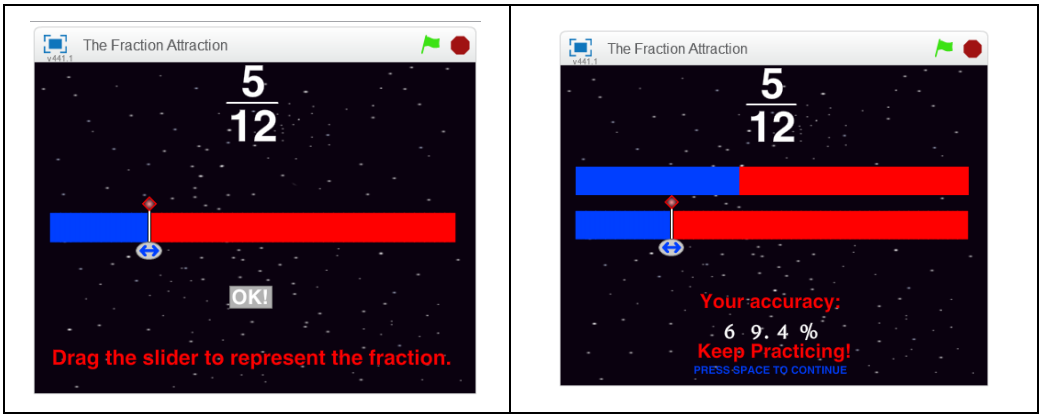
Ici, Scratch est utilisé pour créer un contexte ludique. L'élève-utilisateur ne fait pas d'algorithmique ou de programmation : il utilise un programme tout prêt.

L'activité espérée est : l'élève-utilisateur calcule les fractions.

On peut noter que, vu le design, il est probable que ce soit un élève qui a fait le programme. Si c'est le cas, cet élève-concepteur a, lui, fait de la programmation (mais il a passé plus de temps à gérer le labyrinthe qu'à travailler sur les fractions : le programme est composé de 33 objets et de 238 scripts !).

Une simulation (prise sur le site de la communauté Scratch

Le principe de cette simulation est que l'élève doit faire glisser une réglette pour indiquer à combien il estime une grandeur (ici, 5/12). Le second écran met en évidence la différence entre la solution de l'élève et la solution exacte.



Ici, Scratch est utilisé pour créer un contexte ludique et comme une simulation : le programme apporte à l'élève une information (visualisation de grandeur) dont on peut penser qu'elle a un effet en terme d'apprentissage.

L'activité espérée est : l'élève-utilisateur calcule les fractions et identifie un ordre de grandeur.

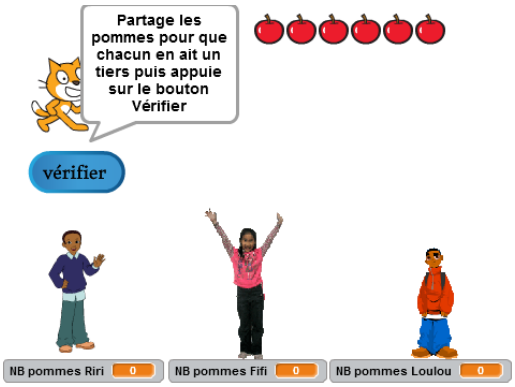
Ici encore, l'élève ne fait pas d'algorithmique ou de programmation, il utilise un programme tout prêt (sauf l'élève qui a fait le programme, si c'est un élève).

Un exercice d'algorithmique/programmation (que j'ai construit sur la base d'exercices de maths de CM1/CM2)

L'idée générale est de proposer aux élèves de *faire* un programme qui pose des questions à l'utilisateur et indique si la réponse est bonne ou pas.

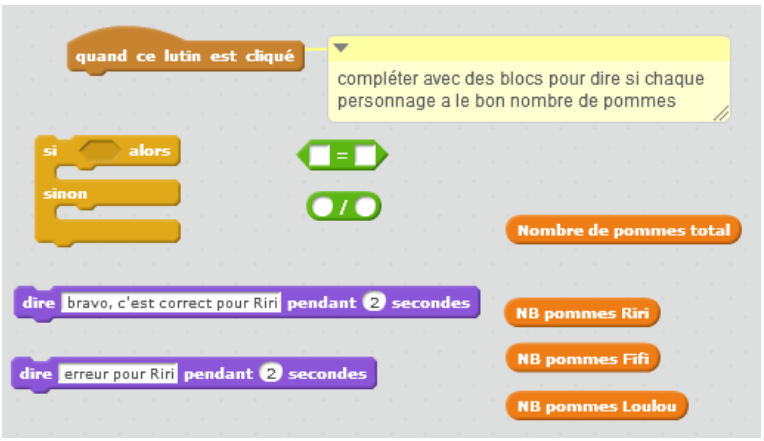
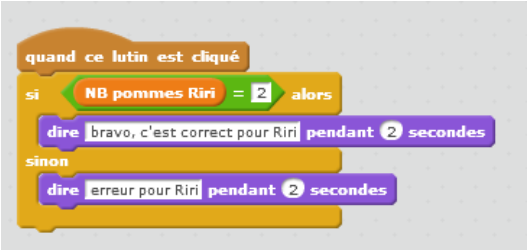
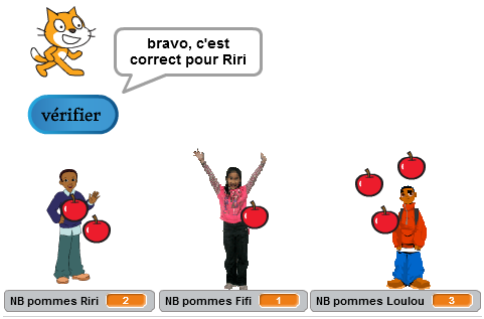
L'architecture générale du programme est prédéfinie : il y a 3 personnages (Riri, Fifi et Loulou), des données (nombre de pommes total, nombre de pommes de chacun) et des scripts (des bouts de programmes pas montrés à l'élève) pour calculer le nombre de pommes de chacun (quand on approche une pomme d'un personnage, son nombre de pommes augmente de 1).

Le personnage « chat » a un script (un bout de programme) qui dit comment il faut répartir les pommes (dit autrement : il affiche un texte décrivant la consigne).



La scène et les deux objets associés à un bout de programme : le personnage du chat (dont le bout de programme, qui vient d'être lancé, indique la consigne) et le bouton « vérifier ».

La tâche-élève est de définir le script du bouton « vérifier », i.e., le bout de programme qui vérifie si les pommes sont bien réparties (si la consigne est respectée) et affiche le résultat à l'écran. Pour cela, l'élève doit écrire un bout d'algorithme de comparaison entre le nombre de pommes de chaque personnage et la bonne réponse. Dans l'exemple illustré ci-dessous, il est aidé par le fait qu'on lui propose un ensemble d'outils (conceptuel et techniques) pour écrire le programme : une structure « si-alors », des opérateurs de comparaison, et les données utiles (le nombre de pommes de chacun).

	La tâche-élève : définir l'algorithme du bouton « vérifier », i.e., les actions qui vont être lancées lorsque l'on va « cliquer ce lutin[42] » (lorsque l'on va cliquer sur le bouton « vérifier »).
	Un début de solution pour Riri (le programme complet doit tester également le nombre de pommes de Fifi et Loulou).
	Le résultat de l'exécution du script ci-dessus une fois réparties les pommes.

L'activité espérée a ici une double dimension. D'une part, l'élève-programmeur doit calculer une fraction (il faut déterminer quand le résultat est correct ou pas pour pouvoir afficher le bon message ; ici, c'est en fait un partage). D'autre part, il doit écrire une séquence d'actions permettant d'afficher le bon message en fonction de la situation, ce qui nécessite d'écrire une instruction conditionnelle (Si-Alors-Sinon) impliquant une expression logique (simple égalité dans ce cas). Dit autrement, dans cet exercice, l'élève fait des maths (pas bien compliqués !) et de l'informatique (algorithmique et programmation). Il est ensuite possible de faire utiliser le programme à un autre élève (un l'élève-utilisateur) qui, lui, ne fera que des maths (répartition des pommes, correcte ou incorrecte).

Une remarque en passant : dans cet exercice l'élève utilise (mais ne crée pas) des variables (« NB pommes Riri », « nombre de pommes total »).

La situation (ici, triviale) permet différentes variantes. Les variantes peuvent permettre de jouer sur différents registres, par exemple :

- Difficulté mathématique : modifier le nombre de pommes en jeu (et/ou trouver un habillage pédagogique pour un exercice de même type mais impliquant des fractions proprement dites), demander une répartition non-uniforme entre les personnages, etc.
- Notions informatiques à mobiliser : reposer la question en cas d'échec (notion de boucle), compter le nombre d'essais (utilisation d'une variable qui va servir de compteur), etc.
- Support aux élèves : aider/guider les élèves (comme ici, en présentant les types de blocs utiles, ou en présentant tous les blocs utiles) ou pas.

[42] En Scratch, les personnages mais aussi tous les objets à qui on associe un bout de programme (comme ici, le bouton « vérifier ») s'appellent des « lutins » (« sprite en anglais »). La notion de « lutin » va bien quand il s'agit d'un personnage, mais elle devient confusionnante quand l'objet n'est pas un personnage.

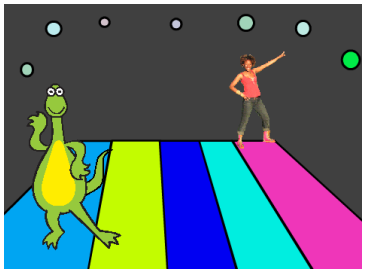
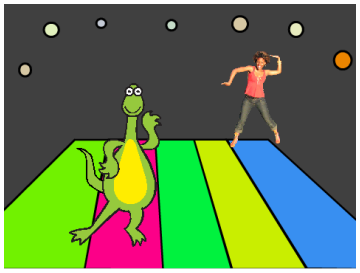
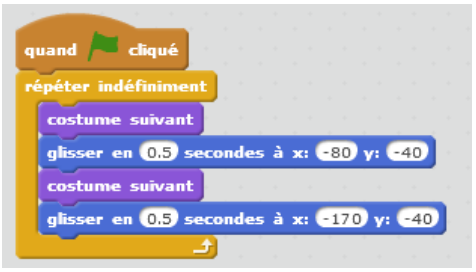
- Niveau d'abstraction : faire travailler les élèves à un niveau instancié (solution = comparaison du nombre de pommes de chacun et de la solution, ici 2) ou/puis plus abstrait (solution = comparaison avec la division du nombre de pommes par le nombre de personnages).

Selon les variantes, l'activité espérée va aborder plus ou moins de notions, être plus ou moins complexe et/ou se situer à différents niveaux d'abstraction[43].

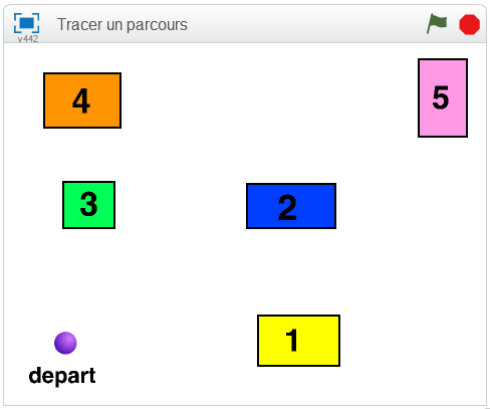
Pour information, j'ai testé cet exercice avec des élèves de CM2 (situation = 3 élèves et moi comme tuteur, avec 3 groupes différents). Ils ont tous immédiatement compris l'utilisation du Si-Alors-Sinon. Une fois la condition écrite pour Riri, à la question posée « et comment on fait pour Fifi maintenant ? », la réponse était bien sûr « on fait pareil » (processus d'abstraction). Et, dès qu'ils ont vu que l'on pouvait dupliquer le bloc en cliquant dessus, ils l'ont dupliqué 2 fois et ont changé les variables et textes (nombre de pomme de Fifi, etc.). Tous ont écrits une condition logique de type « NB pomme Riri = 2 ». Avec des questions comme « et s'il y avait plus de pommes » ou « mais c'est quoi le calcul que l'on fait » certains ont proposé « il faut diviser le nombre de pommes total par 3 » (début de processus d'abstraction), et ont su l'écrire immédiatement. Ils ont utilisé les variables prédéfinies (« NB pomme Riri », etc.) sans hésitation (ce qui ne veut pas dire qu'ils ont compris la *notion* de variable). Dans ces conditions très propices (3 élèves et un tuteur, « bonne classe »), l'exercice a duré ~10-15 minutes. Ceci est « pour information » car ce n'est en rien une expérimentation scientifique contrôlée, c'est juste un test exploratoire.

5.3. Des exemples ludiques

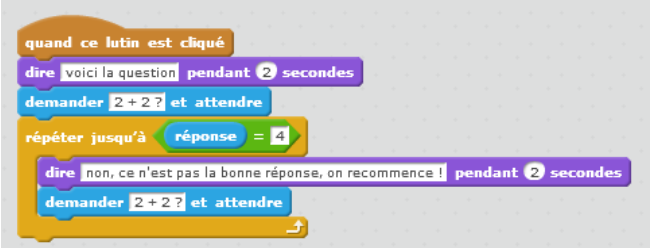
L'exemple classique est de faire danser des personnages en écrivant des boucles. Ci-dessous, une mise en œuvre récupérée sur le site Scratch. C'est ici basique, mais il est possible de faire travailler des compétences assez pointues avec ce type d'habillage pédagogique.

<table>
<tr>
<td colspan="2">   </td>
<td> 
</td>
</tr>
<tr>
<td colspan="2">Les 2 personnages et la scène peuvent être associés à des scripts qui les font bouger, changer de « costume » (en Scratch, un costume est une image du personnage ; cf. le dinosaure, il y a 2 images différentes et on passe de l'une à l'autre), jouer des sons, etc.</td>
<td>Le script du dinosaure : il répète indéfiniment le fait de changer de costume et de se déplacer en glissant dans un sens puis dans l'autre (x et y correspondent aux coordonnées sur la scène, le centre de la scène correspondant aux coordonnées x=0 et y=0)</td>
</tr>
</table>

Autres exemples :

<table>
<tr>
<td> 
</td>
<td>Ecrire le programme qui va faire passer la boule par toutes les cases (compétences mises en jeu : orientation, tracé de segments, angles éventuellement, etc.). On peut faire avant un exercice similaire mais « débranché » dans la cour par exemple.</td>
</tr>
</table>

[43] Pour ceux qui veulent pousser plus loin, une autre variante possible est que les données et/ou la consigne soient produites automatiquement (par exemple, en générant des valeurs aléatoires).

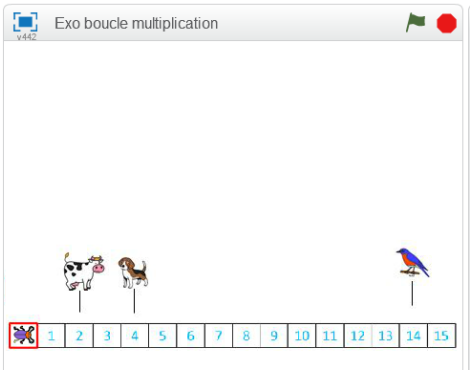
Ecrire un programme qui propose des jeux de question/réponse (en faisant répéter la question tant que la réponse n'est pas bonne ; en comptant le nombre d'essais ; en enchainant plusieurs questions ; en choisissant la question suivante en fonction des succès/échecs précédents ; etc.)	

5.4. Un exemple pour montrer comment aligner le moyen « Scratch » et l'analyse pédagogique/didactique

Les exemples précédents illustrent comment, suite à une analyse pédagogique et/ou didactique, créer une situation où l'élève se voit proposer un ensemble de moyens permettant de produire une solution : une structure Si-Alors-Sinon, une boucle, des opérateurs de calculs, des instructions/blocs permettant de faire dire quelque chose à un personnage, etc. Tous ces moyens sont directement proposés par Scratch.

Il est possible que l'analyse pédagogique et/ou didactique conduise à vouloir proposer aux élèves des moyens qui n'existent pas, mais qui peuvent être construits. Voici un exemple.

Soit l'objectif d'amener les élèves au concept de multiplication en les faisant travailler sur des multiplications par additions réitérées. Un exemple de mise en œuvre est un exercice où il faut écrire un programme faisant avancer un personnage (ici, un insecte) de plusieurs pas pour aller d'un point de départ à un point d'arrivée (la vache, le chien, l'oiseau).

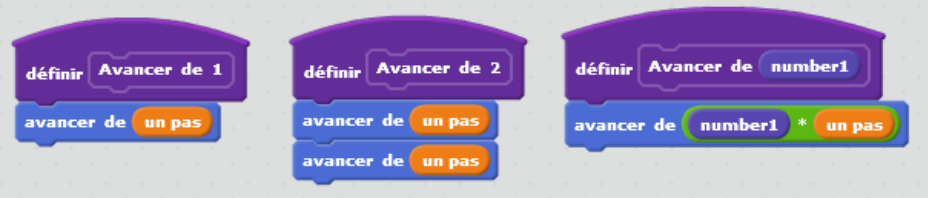
	La tâche-élève est d'écrire les 3 bouts de programmes faisant avancer l'insecte (sur la ligne de départ) vers la vache, le chien et l'oiseau.

Pour ce type d'exercice, les travaux en didactique des maths identifient deux variables didactiques[44] : le nombre de pas et la valeur du pas. Par exemple, la variable didactique « valeur du pas » peut être fixée en proposant une action « Avancer de 1 ». Dans ce cas, pour compléter le script « Aller jusqu'à la vache », l'élève doit mettre trois blocs « Avancer de 1 » (additions réitérées). Il peut procéder de même pour le script « Aller jusqu'au chien », mais cela devient fastidieux pour le script « Aller jusqu'à l'oiseau», où il faudrait juxtaposer 14 « Avancer de 1 ». Ce peut être le ressort pour changer de stratégie et écrire une boucle (« « Répéter 14 fois » : « Avancer de 1 » »). Il est également possible de faire varier la variable didactique « valeur du pas » et de proposer des « Avancer de 1 », des « Avancer de 2 » et « Avancer de 3 » puis, ensuite, un « Avancer de x », x étant une valeur à saisir. L'exercice peut donc être conduit en proposant aux élèves différentes ressources, simultanément ou par étapes.

[44] Pour mémoire, une variable didactique est une variable dont le changement de valeur est sensé modifier la connaissance nécessaire à la solution et susciter une adaptation de stratégie.

Les différents « Avancer » (ainsi, d'ailleurs, que le « Aller sur la ligne de départ ») ne sont pas des moyens directement proposés par Scratch. Ce sont des moyens que j'ai construits, moi, pour les besoins de cet exercice, en utilisant la possibilité de créer des blocs (dit en termes informatiques : de créer des « procédures »).



L'idée qu'il faut retenir est que, si l'analyse pédagogique ou didactique le nécessite, il est possible de créer des blocs qui permettent de proposer aux élèves des moyens ad hoc (mais ce n'est pas forcément à l'enseignant de le faire).

Dans cet exercice, l'activité espérée des élèves est qu'ils écrivent une structure algorithmique de boucle et fassent le lien entre les processus de multiplication et de répétition. J'ai testé cet exercice avec des élèves de CM2 (même situation que précédemment), comme premier exercice car c'est, du point de vue des maths, de niveau CE. L'utilisation du « Répéter » pour aller jusqu'à l'oiseau a ici encore été immédiate. L'exercice a duré ~10-15 minutes.

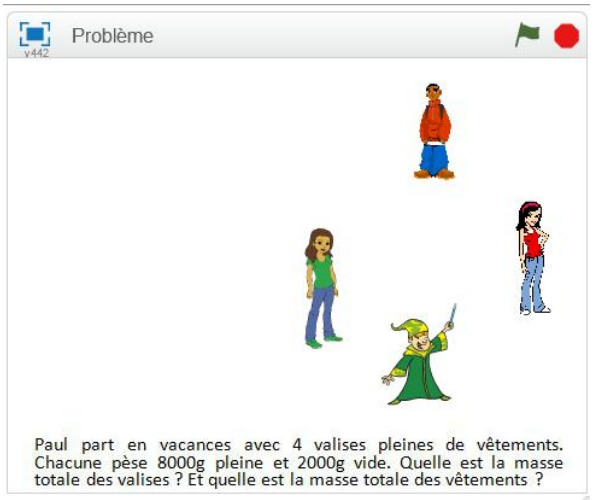
Une remarque en passant : ce type d'exercice fait le lien entre trois compétences (addition, multiplication et boucle) qui peuvent être acquises et/ou en cours d'acquisition. Si l'idée est de s'appuyer sur l'une pour faire acquérir une autre, il faut faire attention au niveau d'acquisition effectif.

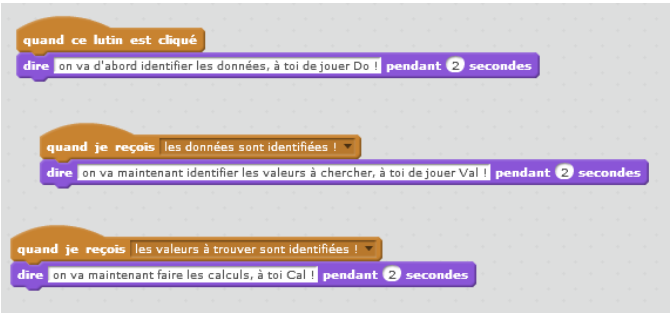
5.5. Un exemple pour montrer comment utiliser Scratch pour scénariser une situation pédagogique

Scratch peut être exploité pour amener les élèves à suivre un scénario pédagogique. Voici un exemple (plus complexe), ici encore inspiré d'un exercice de maths « papier/crayon ».

L'idée est de faire travailler les élèves à la résolution d'un problème. Le comportement espéré des élèves est de suivre la démarche : identifier les données du problème, puis identifier les valeurs que l'on cherche, puis calculer ces valeurs.

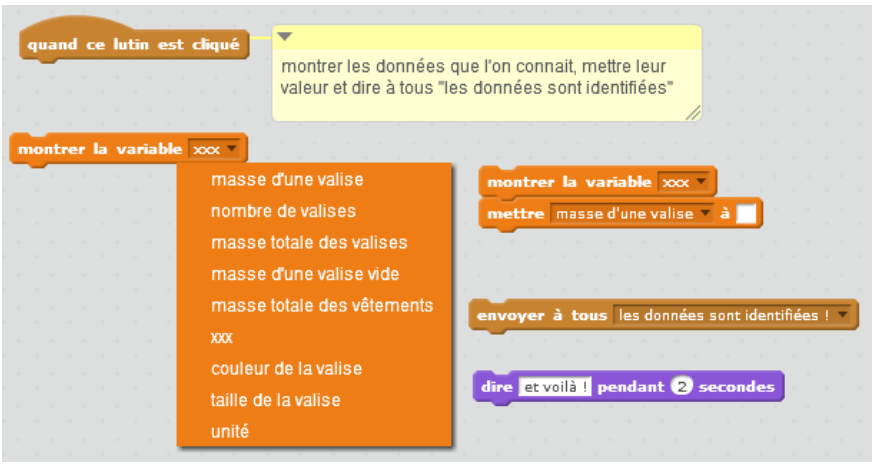
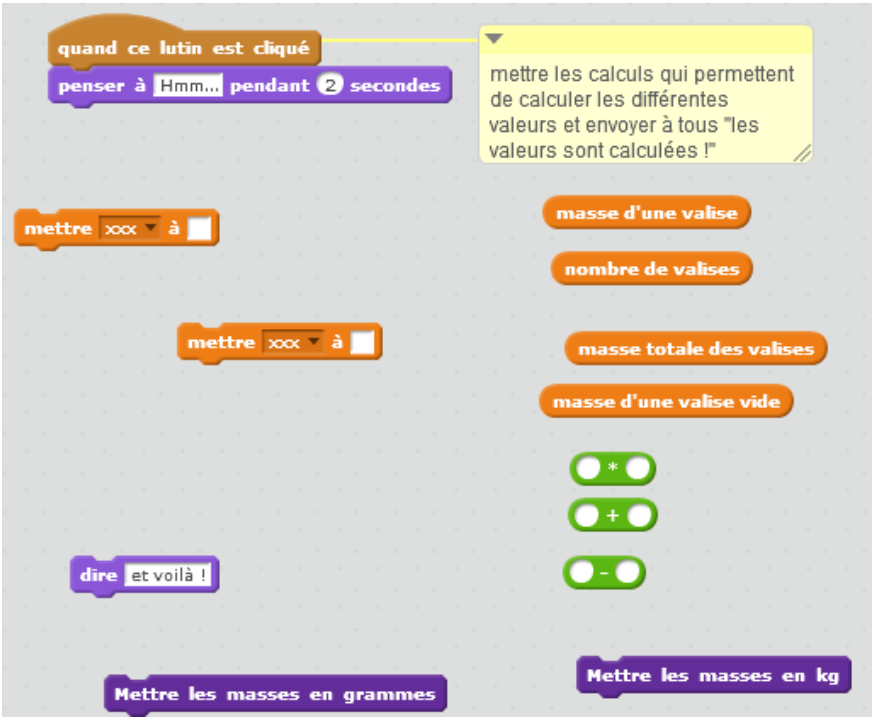
Scratch peut être utilisé pour réifier ce scénario de différentes façons, dont celle-ci : créer 3 personnages (Do, qui est chargée d'identifier les données, Val, qui est chargé d'identifier les valeurs à trouver, et Cal, qui est chargée de faire les calculs). Le 4ième personnage, magicien / chef d'orchestre, définit le scénario : il faut invoquer Do, puis Val, puis Cal. La tâche-élève va être de réaliser les scripts qui sont associés à Do, Val, et Cal.





Le script du magicien / chef d'orchestre : il enchaine les 3 tâches. A la seule fin de donner un exemple de ce type, les scripts des 3 personnages sont synchronisés par envoi de message (chacun envoie un message pour signaler qu'il a terminé sa tâche ; on pourrait faire autrement).

Comme dans les exemples précédents, le travail des élèves peut être facilité en, par exemple, créant un ensemble de variables (celles utiles et, éventuellement, des leurres) et/ou en proposant les blocs utiles.

		Dans cette mise en œuvre, l'élève est notamment aidé par le fait qu'il choisit dans une liste de variables (il serait possible de laisser les élèves identifier et créer ces variables, mais cela suppose une bonne pratique déjà).
		Noter l'utilisation d'un commentaire pour indiquer la consigne.
		Noter la proposition de blocs permettant de changer d'unité (Kg, g).

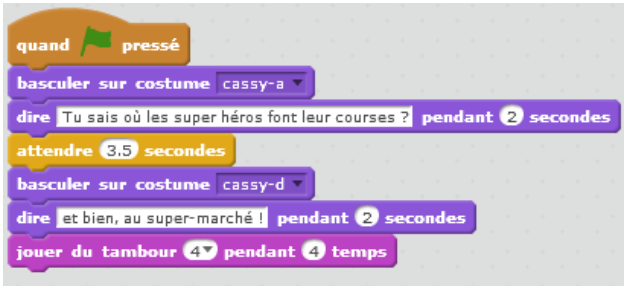
J'ai testé cet exercice avec des élèves de CM2 (même situation que précédement). L'exercice a duré ~15-20 minutes. Exemple de comportement intéressant : au moment de faire les calculs, certains élèves proposent de « multiplier la masse d'une valise par 4 » (i.e., font référence à la variable), d'autres proposent de « faire 8000 que multiplie 4 » (i.e., font référence à la valeur). La notion de variable me semble complexe pour des élèves de CM1/CM2 et, si elle est utilisée, elle doit être introduite avec soin.

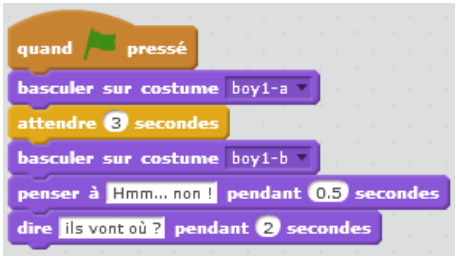
5.6. Un exemple pour montrer comment passer de la production de texte à l'informatique (et inversement)

Soit l'objectif de faire travailler la compétence « production de texte ». Il est possible d'utiliser Scratch pour créer un contexte ludique, par exemple en définissant une tâche (un exercice) du type « soit le contexte (…), imaginer le dialogue entre Alice et Paul et l'animer sur l'écran ».

La tâche-élève peut être facilitée en proposant aux élèves une structure de programme prédéfinie où il y a deux personnages, Alice et Paul, et les actions de Scratch permettant de « faire dire quelque chose » aux personnages. Les élèves n'ont plus qu'à taper les dialogues successifs de ces personnages. On peut éventuellement expliquer comment ajouter des animations (mouvements des personnages, sons, etc.) pour renforcer la dimension ludique et, potentiellement, la motivation, en faisant attention à ce que l'activité des élèves ne devienne pas uniquement ludique (tâche-d'enseignement).

Ici, l'activité espérée des élèves est : écriture de dialogues / composition dramatique (…).





Pour créer un dialogue, il suffit de copier/coller des séquences « attendre / dire », puis de modifier le texte qui est « dit ». Le fait d'utiliser des personnages ayant plusieurs « costumes » dénotant différentes émotions peut être un vecteur pour faire développer un dialogue complexe (consigne = utiliser les costumes dénotant la surprise, la réflexion, etc.).

Reprenons le même exercice en ajoutant : faire saisir des textes à l'utilisateur et faire en sorte que Alice et Paul disent des choses selon ce que tape l'utilisateur (par exemple : son âge, ses préférences sur un sujet donné, etc.) ; variante : l'utilisateur agit sur la situation en cliquant sur un objet, appuyant sur une flèche du clavier ou en bougeant la souris. Cet exercice va maintenant amener à utiliser la notion algorithmique de conditionnelle Si-Alors-Sinon et à l'écriture d'expressions logiques, et éventuellement à définir/utiliser des variables. La création de situations avec plusieurs personnages amène par ailleurs assez rapidement à dépasser une synchronisation simpliste (en utilisant des « Attendre x secondes ») pour utiliser des envois de messages par exemple.

Complétons maintenant en ajoutant : les élèves doivent réaliser une histoire (« fiction interactive »), ou un jeu. Il faut alors qu'ils construisent un schéma narratif (situation initiale, présentation des personnages, lieu, but, éléments déclencheurs, déroulement, situation finale) ; détailler chaque scène (personnages/lutins, scènes, arrière plans, décors, sons, comportements des objets, etc.) ; créer des consignes, des dialogues, des aides ; inventer des énigmes à résoudre (et créer les bouts de programme permettant de tester les réponses de l'utilisateur !) ; etc. Tout ceci peut se faire en travaillant papier-crayon tout d'abord (ce peut être à différents niveaux de précision, de l'idée générale à un scénario ultra-précis) puis sur machine, ou directement sur machine. C'est bien évidemment un projet sur plusieurs séances, pour lequel il faut construire une organisation (cf. remarque / technique de gestion de projet en Section 4.3).

5.7. Discussion

Les exemples utilisés dans cette section relèvent du schéma prototypique consistant à proposer aux élèves une tâche du type « faire un programme qui fait xxx », le fait de faire ce programme nécessitant de mobiliser les compétences cibles : raisonner en termes de décomposition du problème en sous-problèmes, écriture d'un algorithme, écrire une boucle, etc. La tâche proposée peut varier d'un exercice très spécifique (par exemple, l'objectif étant de faire travailler la notion de conditionnelle, les faire compléter un programme qui nécessite un Si-Alors-Sinon) a un projet nécessitant une démarche complète (le fameux « jeu de Pong » : analyse du problème, élaboration d'une idée générale, création des algorithmes, codage, tests). L'exercice peut consister à compléter une structure de programmes existante (j'ai privilégié ce type d'exemple car il permet de préciser/structurer la tâche élève) ou créer un programme ex nihilo. L'informative créative relève, dans sa dimension algorithmique, du cas limite où les élèves définissent eux-mêmes le problème qu'ils considèrent.

Dans ce contexte, Scratch peut être introduit après une phase papier/crayon (écriture d'un plan général puis d'un algorithme précis en langage naturel), comme un moyen de représenter l'algorithme produit et de constater, par

l'exécution du programme, si la solution proposée est satisfaisante ou pas (idée = considérer d'abord des notions / un langage abstrait, e.g., la notion de boucle, puis sa traduction en Scratch). Scratch peut également être utilisé directement, sans phase papier-crayon ou après une analyse très générale, comme un support pour « penser » en termes algorithmiques (exploitation du fait que Scratch réifie les structures algorithmiques par des blocs visuels, cf. les exemples précédents). L'exécution au fur et à mesure de bouts de programmes permet de tester/évaluer les idées sur le mode essai-erreur.

Du point de vue des compétences générales à aborder au primaire, la réalisation d'un programme peut donc être vue comme une instance du triptyque anticipation / contrôle / validation.

Que l'on passe par une étape abstraite ou pas, il faut amener les élèves à savoir identifier la nature de ce qu'ils veulent faire faire au programme puis à écrire les choses précisément : c'est le passage de « il y a une répétition » à « on va répéter telle action jusqu'à ce que telle condition soit vérifiée ». A ce niveau, par rapport à des langages standards, Scratch propose une aide très importante : décider de réaliser une action correspond à chercher dans la liste proposée le « bloc » qui semble le plus adapté. Dit en d'autres termes, l'élève est en situation de sélection.

Dans le cas où l'on introduit un niveau préliminaire abstrait, il faut faire attention à la correspondance entre le langage « abstrait » avec la façon dont Scratch structure les moyens qu'il propose (types de blocs « évènement », « contrôle », « mouvements », etc.). Les auteurs de Scratch ont choisi de structurer les choses d'une certaine façon, on peut éventuellement penser que cela aurait été mieux autrement … mais on ne peut pas la changer. Il faut faire attention à ne pas amener à des confusions et/ou à ce que les élèves passent beaucoup de temps à chercher au pifomètre dans les blocs, ce qui serait contre-productif.

Il faut enfin amener les élèves à savoir organiser la solution, i.e., écrire l'algorithme complet, ce qui en Scratch correspond à agencer les blocs pour former un ou des scripts. Ici encore, par rapport à des langages standards, Scratch propose une aide très importante : cela se fait par des mouvements d'objets sur l'écran, et le système interdit les constructions syntaxiquement incohérentes.

Il est important de noter qu'identifier les notions/moyens à mobiliser pour résoudre le problème considéré ne veut pas dire savoir produire le programme d'un trait. Il y a une compétence qui consiste à penser/définir la stratégie générale (l'algorithme général), i.e., décrire en langage naturel, de façon plus ou moins précise et structurée, les actions que doit réaliser le programme. Et il y a d'autres compétences qui consistent à savoir consulter les moyens précis offerts par le langage utilisé, décider lequel semble le plus adapté, tester et évaluer, adapter/modifier. C'est ceci qui justifie la remarque indiquant qu'il n'est pas nécessaire de maîtriser parfaitement le langage de programmation utilisé au sens de « connaître tous les détails » ou « savoir résoudre tous les problèmes instantanément ».

Un point clé, évident mais qu'il faut répéter : en tant qu'enseignant, bien réfléchir à ce que l'on fait soi et à ce que font les élèves (ne pas tomber dans la situation bien connue où l'élève regarde l'enseignant résoudre le problème, en l'occurrence regarde l'enseignant faire le programme).

Une question enfin : que font les élèves quand ils font des exercices impliquant un autre domaine (par exemple, les maths) du type de ceux présentés ci-dessus ? Des maths ? Des maths dans un contexte informatique ? De l'informatique dans un contexte de maths ? Des maths et de l'informatique en même temps ?

6. Que faut-il comprendre de Scratch en tant qu'enseignant, et comment s'y prendre ?

Ainsi qu'indiqué précédemment, l'un des écueils de l'enseignement de l'informatique au primaire est que les enseignants concernés n'ont, pour certains, jamais étudié l'informatique dans leur parcours de formation initiale[45]. Encore une fois, ceci ne me semble pas rédhibitoire, ni même un désavantage majeur. En revanche, par manque d'expérience personnelle, cela peut amener à des difficultés à percevoir ce que signifie « être assez compétent pour enseigner ». Dans cette section j'essaie de donner quelques éléments de réponse pragmatiques (sans aborder la question, par ailleurs légitime, de la certification).

[45] Les enseignants actuellement en formation et qui vont enseigner l'informatique et la programmation en collège et lycée sont (probablement) ceux des matières scientifiques, qui viennent donc de parcours de maths physique etc. et ont donc tous déjà appris et pratiqué la programmation « standard » dans leur licence/master. Ceci étant, il serait sans doute intéressant que les enseignants des différentes disciplines développent une certaine familiarisation avec l'enseignement de l'informatique (par exemple, du niveau évoqué par le « socle commun » ?), ne serait-ce que pour qu'ils puissent faire le lien avec leur discipline lorsque c'est pertinent et, éventuellement, considérer des enseignements interdisciplinaires.

6.1. Quels objectifs de compréhension/connaissance du langage se fixer ?

Pour enseigner la pensée informatique, il faut une compréhension de Scratch en tant que langage informatique, i.e., en tant que moyen pour exprimer une solution et, plus spécifiquement, un algorithme (liste des actions à réaliser pour que le programme fasse ce que l'on souhaite).

L'un des enjeux (tant pour l'enseignant que pour les élèves) est de savoir dissocier les possibilités du langage qui sont des moyens pour réaliser la tâche et les choses contingentes (par exemple, les possibilités de dessiner des personnages, de jouer des sons, etc.). Attention cependant : des choses contingentes étant donné un objectif pédagogique ou une tâche peuvent être centrales pour une autre, par exemple les aspects esthétiques pour le développement de la créativité. Des choses contingentes peuvent, par ailleurs, être des éléments essentiels pour (par exemple) entretenir la motivation.

Pour certains objectifs pédagogiques, le fait de « savoir utiliser l'environnement » peut être suffisant. Par exemple, si l'on reprend l'objectif de faire travailler la production de textes, il est possible de juste savoir comment faire dire quelque chose à un personnage, de réutiliser un script qui synchronise les échanges, et de faire travailler les élèves sur les textes de ces échanges.

6.2. Comment s'y prendre ?

Première chose à faire, pour avoir une idée intuitive des choses : aller sur le site Scratch[46] (environnement en ligne, téléchargement logiciel pour utilisation locale, communauté, exemples, etc.) ; lancer quelques programmes et en modifier quelques détails, en utilisant l'aide en ligne et les tutoriels éventuellement. Une propriété très importante de Scratch est de permettre de « voir à l'intérieur » de tous les programmes, i.e., voir comment ils sont réalisés. Ceci permet notamment de découvrir les notions, l'organisation de l'interface (etc.) en regardant quelques programmes, en les dépiautant et en les modifiant. Plus généralement, cela permet de réutiliser facilement des programmes ou bouts de programmes existants[47].

Ensuite, pour développer une connaissance de Scratch, deux références essentielles (on trouve par ailleurs d'innombrables présentations sur le Web) :

- Le document « Concepts de programmation et compétences développés avec Scratch »[48], qui synthétise en 2 pages les structures algorithmiques de base, quelques autres notions (variables, synchronisation, etc.) et comment les représenter en Scratch. Ce document est utile pour faire le lien algorithmique/Scratch.
- Le guide référence (23 pages)[49], qui détaille l'environnement et le langage (les différents menus, la scène, les personnages et leurs costumes, etc.). Ce document est utile pour savoir utiliser Scratch.

Autre approche, complémentaire ou alternative : travailler sur les manuels pédagogiques associés à Scratch (cf. Section 7).

Comme indiqué précédemment, il n'y a pas besoin d'être hyper-pointu, de connaitre le langage par cœur et de savoir tout faire. Cependant, le risque d'éluder cette phase de travail sur ces documents est de ne pas comprendre et/ou de devoir se limiter à réutiliser les exemples existants, i.e., de ne pas maitriser son enseignement et ne pas être capable d'utiliser l'espace des possibles qu'ouvre Scratch. Et, par ailleurs, de ne pas savoir où trouver rapidement la réponse à une question d'un élève.

Ainsi qu'indiqué précédemment, je pense que c'est sans doute une bonne idée que l'enseignant se positionne auprès des élèves comme en termes de « je ne sais pas tout du langage », et que la situation « je ne sais pas, on va chercher ensemble » soit banalisée. Cela renforce la différence entre la compétence qui consiste à penser la solution et celle qui consiste à la traduire dans un langage technique, et, au niveau du langage technique, entre la connaissance des structures de base (bloc « répéter », bloc « si-alors-sinon ») et les actions/possibilités propres au langage Scratch (« faire avancer le personnage jusqu'au bord de la fenêtre et rebondir », « gérer les costumes » etc.). L'informatique, ce n'est pas connaitre des langages de programmation par cœur. L'enseignant, comme l'élève, comme l'informaticien souvent, doit être capable de, si/quand nécessaire, « consulter la doc » (dans Scratch : regarder la liste

[46] https://scratch.mit.edu/ ; voir aussi http://scratchfr.free.fr/

[47] Attention cependant : le site Scratch propose (au moment où j'écris ce document) plus de 12 millions de projets. L'immense majorité sont sans intérêt, ni ludique ni pédagogique. Il ne faut pas s'arrêter au premier exemple venu donc.

[48] scratchfr.free.fr/j8y3r7/pc1.4fr070109A4.pdf

[49] http://scratchfr.free.fr/k1n8g7/ScratchRefGuidefrv14A4.pdf

des blocs et choisir le plus approprié et/ou lire l'aide proposée par l'environnement et/ou consulter des programmes qui font ce que l'on voudrait faire faire au notre et/ou lire la documentation).

Scratch peut être utilisé en ligne ou téléchargé sur un ordinateur et utilisé localement. Il existe plusieurs versions (1.4 et 2 actuellement) et des adjonctions (pour piloter des constructions électroniques par exemple). Les différences ne semblent pas être un enjeu important pour le primaire et on peut donc utiliser la version la plus pratique dans le contexte local (connexion Internet, etc.).

6.3. Quelques remarques

Quelques remarques personnelles sur les choses importantes à connaître/comprendre, sans reprendre ce qui est décrit et bien décrit dans les documents mentionnés :

- La métaphore : une scène de théâtre ; sur la scène, des lutins (des personnages en général) qui jouent (qui font des choses) ; ce que fait un lutin est défini par ses scripts, qui sont constitués de blocs ; un lutin peut avoir plusieurs costumes, une scène peut avoir plusieurs arrière-plans, lutins et scène peuvent être associés à des sons.
- Les concepts sous-jacents (lutins, scène, scripts, costumes, etc.). Ce ne sont pas tous des concepts informatiques, ils ne sont pas tous (ou tout le temps) utiles, ils sont sur certains aspects contingents et/ou discutables. Mais c'est justement pour cela que l'enseignant doit les connaitre, pour pouvoir faire la part des choses (savoir-faire cette part des choses est sans doute l'un des enjeux importants de la formation).
- Les notions/mécanismes de base que l'on peut cibler au cycle 3 : cf. Section 3.2. Mais une autre façon de voir les choses est de définir un projet et d'enseigner les notions utiles (centration sur le projet mené et pas sur le domaine d'enseignement).
- Un truc tout bête mais important : la possibilité de dupliquer des blocs par un click droit (travail sur l'abstraction et la réutilisation).
- Un autre truc tout bête mais qu'il serait dommage de ne pas utiliser : la possibilité d'associer des « commentaires » (des petits textes) aux scripts, ce qui permet de produire une description abstraite, justifier ses choix, documenter, etc.
- Une propriété de Scratch est qu'il est possible de lancer une instruction ou une séquence d'instructions en cliquant dessus. Cela permet de faire des tests sur le mode essai/erreur. Ce peut être également un moyen d'aider les élèves à élaborer la solution. Par exemple, voir ce qu'il se passe quand on clique sur le « Avancer de 1 » de l'insecte, voir ce qu'il se passe quand on clique sur « Aller sur la ligne de départ », voir ce qu'il se passe quand on lie les deux, etc.
- Le fait de savoir définir des blocs simples (comme les « Avancer » de l'exemple de l'insecte) est un objectif facilement atteignable une fois familiarisé avec l'environnement et ses notions. Mais ce n'est pas une condition nécessaire pour utiliser Scratch.

7. Comment définir et gérer des situations pédagogiques ?

En l'absence de programme d'enseignement précis, il est difficile d'expliquer comment il faut définir les situations pédagogiques car cela dépend beaucoup du contexte dans lequel l'informatique/Scratch est abordé : enseignement en tant que tel, prévu et organisé sur plusieurs séances/séquences, ou exercices ponctuels ? Pédagogie par projet ? Utilisation banalisée de Scratch comme média support à la créativité ? Lien avec les autres activités de la classe ? Nombre d'heures ? Etc. Cette section est donc assez générale et ne fait qu'indiquer des pistes.

7.1. Exploiter les manuels autours de Scratch

Il existe des manuels dont, notamment, disponibles sur le Web :

- Bien commencer avec Scratch, introduction à l'informatique (2013)[50], traduction Française du livre « Starting from Scratch: an introduction to computing science ».

[50] https://pixees.fr/?p=3372

- L'informatique créative (2011 puis 2013 ?)[51], traduction Française du livre « Creative Computing Curriculum Guide ».

Ce sont, dans les deux cas, des manuels qui insistent beaucoup sur la dimension créativité. Tous les deux proposent des manuels enseignants et manuels élèves.

Le titre de « Bien commencer avec Scratch, introduction à l'informatique » dénote bien l'objet du livre : faire découvrir l'informatique *à travers* l'apprentissage du langage Scratch. Logiquement, le livre commence par proposer des activités pédagogiques de découverte de l'environnement et du langage, puis des activités pédagogiques construites autour de tâche-élèves du type « créer un jeu », « créer des figures », etc. Chaque activité est décrite en termes de : concepts introduits, qui mélangent des notions informatiques (e.g., conditionnelle, boucle, variable) et autres (e.g., « création d'un jeu », « le chronomètre ») ; les commandes Scratch introduites ; le lien avec les compétences relevant de la pensée informatique (abstraction, algorithmes, décomposition de problème en sous-problème, reconnaissance de formes, généralisation) ; les objectifs (ce que l'élève devrait être capable de d'avoir compris après l'activité) et les ressources utilisées.

Le titre de « L'informatique créative » dénote également bien l'objet du livre : il est centré sur la dimension créative. On y trouve des descriptions d'activités qui insistent sur ce type d'objectif (e.g., « A la fin de cette activité, les élèves auront pu exprimer leur créativité en relevant un défi ayant une thématique artistique »), objectifs qui sont mélangés avec des objectifs de maitrise du langage Scratch (e.g., maitriser les blocs de la catégorie « Apparence » et l'éditeur d'images) et des objectifs de maitrise de notions informatiques (e.g., la notion de répétition/boucle). La dimension algorithmique est moins prégnante que dans le premier manuel.

Ce type de manuel peut être utilisé, très directement, pour faire travailler les élèves sur tout ou partie des activités proposées, dans leur logique donc. Typiquement, commencer par les premières activités qui visent à familiariser les élèves avec l'environnement, puis piocher dans les autres activités. Ils explicitent quelques éléments de la « tâche-d'enseignement » : « importance de donner l'occasion aux élèves de parler de leurs créations et de leurs pratiques créatives », stratégies d'évaluation, proposition de progressions, activités de consolidation.

Il existe bien sûr d'autres ressources, par exemples les « cartes Scratch[52] ».

7.2. Construire des situations pédagogiques

Une autre façon de construire des situations pédagogiques est de partir d'objectifs d'apprentissage précis et de construire, sélectionner et/ou adapter des exercices. Par exemple :

- Partir de ressources produites pour d'autres disciplines (cf. exemple des fractions en Section 5.2).
- Partir d'analyses didactiques (cf. l'exemple de la numération en Section 5.4).
- Reprendre ou s'inspirer des activités des manuels Scratch qui mobilisent la notion/compétence cible.
- Partir sur des projets identifiés pour leur capacité à mobiliser les élèves, et les définir et les conduire de façon à amener les élèves à travailler certaines compétences.
- S'appuyer sur des activités/projets liés à l'histoire de la classe.
- …

On peut penser/espérer que, une fois la vague « effet de mode » passée, vont subsister et se développer des banques d'exercices constituées de couples analyse pédagogique/didactique + structures de programme Scratch « à compléter » tout prêts à l'emploi (structures de programme Scratch comprenant les éventuels blocs ad hoc nécessaires, blocs que l'enseignant n'aura donc qu'à comprendre et pas à créer).

8. Quelles difficultés peut-on attendre/anticiper ?

En l'absence d'analyses de situations banalisées (au sens de : qui ne sont pas des expériences de précurseurs enthousiastes) et de travaux didactiques, il est difficile de développer une analyse des difficultés. Je dresse donc ci-dessous une liste de difficultés potentielles[53].

[51] http://scratched.gse.harvard.edu/resources/informatique-créative

[52] http://scratchfr.free.fr/ma02138/PAG_CardsA4Eu033110.pdf

[53] Je remercie par avance les enseignants et/ou lecteurs des contributions qu'ils m'enverront.

- Accès à des ordinateurs. Pour que des élèves deviennent à l'aise et/ou autonomes avec un d'environnement de programmation comme Scratch il faut l'utiliser régulièrement, et donc avoir un accès facile/régulier à des postes de travail. Une propriété importante de Scratch est cependant que l'on peut travailler en ligne (pas besoin d'installer le logiciel sur le poste donc) et l'on peut créer/gérer un compte pour y mettre ses programmes (pas besoin de toujours travailler sur le même ordinateur ou de transférer des données). Le fait de faire travailler les élèves par groupes de 3 ou 4 peut par ailleurs se gérer via des ateliers tournants.
- Tenue de la classe. Les expériences diverses illustrent toutes l'enthousiasme des enfants, ce qui veut également dire … excitation, bataille pour tenir la souris, utilisation de l'environnement à des fins uniquement ludiques, etc. Ceci peut compliquer la tâche d'enseignement (gérer plusieurs groupes travaillant simultanément sur plusieurs postes de travail doit être assez sportif[54]).
- Guidage/autonomisation des élèves. L'environnement Scratch offre une variété de possibilités : créer des personnages, créer des scènes, créer et jouer sur les costumes des personnages, faire se déplacer les personnages, associer des sons, etc. Ainsi qu'indiqué précédemment, c'est lié à sa raison d'être (support à la créativité informatique et pas, simplement, au codage d'algorithmes). Pour arriver à faire travailler les élèves il faut soit être avec eux pour les aider à ne considérer que ce qui est utile pour la tâche en cours (ce qui mobilise l'enseignant donc), soit les rendre autonomes (ce qui prend du temps). Noter que les manuels indiqués en Section 7 proposent des activités destinées à ce que les élèves découvrent/maitrisent l'environnement.
- Relation enseignant/élève. Tout enseignant d'informatique a un jour « séché » devant un élève/étudiant qui lui demandait pourquoi le programme ne faisait pas ce qu'il attendait ou comment on pouvait faire quelque chose. Pour gérer ces aspects, il est important d'introduire l'ordinateur/le langage comme un moyen, qui peut présenter des propriétés que l'on ne connait pas toutes (mais qu'on peut explorer pour revenir, plus tard, avec une explication).
- Charge de travail. Se former, articuler avec les autres enseignements, etc. : c'est du travail en plus, cela peut démotiver et/ou épuiser. A ce niveau, il faut espérer que (1) il va se développer des ressources « prêtes à l'emploi » (banque d'exercices, etc.) et (2) des études scientifiques donneront aux enseignants des bonnes raisons de faire ce type d'enseignement en montrant que l'enseignement de l'informatique fait effectivement progresser les élèves (ou certains élèves) sur certaines compétences plus que (ou différemment) d'autres activités pédagogiques et que, au-delà du côté ludique et des aspects motivationnels associés, c'est une bonne idée de faire ce type d'enseignement.
- Représentations des élèves. Il fut un temps où enseigner l'informatique commençait par montrer un ordinateur et expliquer comment s'en servir. Cela nécessitait d'expliquer des choses de base (comment utiliser un clavier et une souris) mais présentait l'avantage d'introduire des conceptions sur un terrain vierge. Les élèves ont maintenant pour la plupart sinon tous utilisé un ordinateur ou une tablette, et développé des représentations de leurs usages (en général plutôt ludiques bien sûr mais, surtout, comme utilisateur et non comme producteur). La contrepartie positive pourrait être qu'il est donc inutile de leur faire pratiquer l'usage du clavier et de la souris mais ce n'est pas forcément le cas (cf. remarque suivante).
- Problème de motricité. J'ai été surpris de voir certains élèves avoir des difficultés à déplacer des blocs Scratch « à la souris » et, surtout, à avoir des problèmes pour éditer des données/texte (par exemple dans des blocs « Dire xxx » ; la difficulté n'était pas de saisir des textes au clavier, mais de cliquer sur la zone où éditer sans déplacer le bloc). Plusieurs institutrices m'ont confirmé que les élèves de CM1-CM2 ne savaient pas utiliser une souris de façon précise (dans certains cas car … ils utilisent surtout des tablettes tactiles ou des consoles de jeux).
- Problèmes pédagogiques. En l'absence d'expériences/travaux/consensus étayés (à ma connaissance), il y a différentes questions ouvertes dont, notamment :
    - Quelle est la mesure dans laquelle, à l'école élémentaire, on peut (et c'est une bonne idée de chercher à) faire dissocier aux élèves la démarche abstraite, s'appuyant sur des

[54] Une institutrice ayant expérimenté en grand groupe (25 élèves) avec 3 adultes dans la classe m'indique sur le mode de la litote : « cela engendre un bruit et une agitation assez conséquente, je confirme (…) nous sommes trois adultes mais nous sortons de la séance à chaque fois épuisées, car les enfants sont encore moins patients quand ils ont besoin d'aide dans ce genre de travail et on a toujours ce sentiment de "l'ordi-il-veut-pas-faire-ce-que-je-lui-dis"/"maitreeeeeeeeessse, ça "bugue"" ». Cf. le point suivant sur l'automisation.

notions/principes généraux, et sa mise en œuvre avec un outil, en l'occurrence Scratch ? Faire la différence entre sa solution et comment la mettre en œuvre avec un outil (le langage Scratch) est bien sûr fondamental, et l'étape suivante (au collège et/ou lycée) est de généraliser cela par une phase « et je choisis mon outil, i.e., mon langage de programmation, en fonction de ce que je veux/dois faire ». Mais c'est toujours compliqué, quel que soit le niveau d'enseignement d'ailleurs, quand on présente à la fois un problème et un moyen (les élèves savent bien qu'ils vont le « faire en Scratch », il y a un biais dès le départ, est-il nécessaire de passer du temps à faire la différence ? D'autant que les blocs Scratch sont des structures algorithmiques très propres).

- Quelle est la balance entre faire « travailler par réutilisation » et faire « créer son programme » ? Savoir réutiliser est une compétence utile, qui nécessitent des processus non-triviaux : comprendre un problème et une solution ; faire le lien entre deux problèmes et/ou deux solutions (abstraction, reconnaissance de forme/pattern) ; passer de l'abstrait au concret (instancier le pattern, réutiliser et modifier le script réutilisé). Ceci étant, ne savoir procéder que sur le mode « j'ai déjà vu un programme qui fait cela, je le prends et je le modifie » ne garantit en rien une compréhension/maitrise des choses. Savoir réutiliser est une compétence utile … mais pas suffisante, surtout si on réutilise quelque chose que l'on ne comprend pas. Par ailleurs, savoir réutiliser et savoir créer ex nihilo sont deux compétences différentes (c'est un problème bien connu en enseignement de l'informatique, du type « syndrome de la page blanche »). Tout ceci est à considérer en lien avec les objectifs pédagogiques bien sûr.
- Quelle place donner à l'enseignement de la pensée informatique, de l'algorithmique, de l'utilisation de Scratch ? Enseignement en tant que tel, avec et/ou au sein d'un autre domaine disciplinaire ? Utilisation « créative » de Scratch dans le cadre d'activités de délestage (par exemple : réaliser des cartes de vœux plus ou moins animées/interactives), activités ayant par ailleurs vocation à familiariser l'élève avec l'environnement et, du coup, à permettre des séances « orientées informatique » plus focalisées et plus efficaces ? Les équilibres sont sans doute à étudier au cas par cas (équipement informatique de la classe, place donnée à l'informatique / la créativité, histoire et mode de fonctionnement de la classe, etc.).
- Comment aborder et gérer la notion d'erreur ? L'utilisation d'une machine qui permet de constater l'effet de ce que l'on a produit peut être exploité pour amener les élèves à développer des démarches exploratoires, pour favoriser le « tâtonnement expérimental » (anticipation / contrôle / validation). Et, bien sûr, pour déstabiliser des conceptions erronées (tâche d'enseignement). Le pendant moins positif est le risque de démarche essai/erreur sans travail de réflexion.
- Quelles conceptualisations des notions sous-jacentes (y compris celle de « langage de programmation ») développent les élèves ?
- …

9. Conclusion

Dans les commentaires d'enseignants recueillis sur ce document l'un d'eux me rappelait que : « *Les choses que j'ai le plus réutilisées dans mes classes sont celles que j'ai pu expérimenter à l'IUFM en me mettant en situation d'élève puis en discutant avec mes collègues des difficultés, des mises en œuvre en classe avec des séquences et des séances très détaillées* ». Bref : il faut pratiquer.

Le cours dont ce document est le support papier est prévu en 3 phases. Tout d'abord, des présentations/discussions des différents points abordés dans ce document, illustrés et rendus concrets par des manipulations de programmes Scratch. Ensuite, les étudiants-professeurs-des-écoles préparent, chacun ou par groupe, une séance pédagogique (un exercice à destination d'élèves de l'école élémentaire, en explicitant la réflexion pédagogique sous-jacente). Enfin, des présentations puis analyses collectives de ces séances pédagogiques.