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

Version 2 du 25 mars 2017[1]

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 faut donc prendre de la distance par rapport à un effet de mode et des discours parfois trop généraux et/ou trop prosélytes (ou caricaturalement négatifs), dont certains 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 infondés (pour certains, complètement idiots), ou encore porteurs de confusions (par exemple qu'enseigner l'informatique a pour but de faire des élèves de futurs programmeurs : enseigner la lecture et l'écriture ce n'est pas former des écrivains, même si certains élèves deviendront peut-être écrivains). 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 bien 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 connaître et comprendre le domaine, les discours, les programmes et les principales ressources existantes mais, surtout, pour développer une réflexion personnelle.

Comme son titre l'indique, ce document se focalise sur l'enseignement de « la pensée informatique », i.e., les compétences suivantes : savoir décomposer un problème en sous-problèmes plus simples ; savoir réfléchir aux tâches à accomplir pour résoudre un problème en termes d'étapes et d'actions (ce que l'on appelle un algorithme) ; savoir décrire les problèmes et les solutions à différents niveaux d'abstraction, ce qui permet d'identifier des similitudes entre problèmes et, par suite, de pouvoir réutiliser des éléments de solutions ; ainsi que, dans l'approche que j'adopte, savoir écrire et tester ces algorithmes avec le langage de programmation Scratch. Pourquoi Scratch ? Parce que c'est le langage qui s'impose de fait pour le niveau scolaire / la classe d'âge considérée, et qu'il y a de bonnes raisons à cela. L'essentiel du discours est cependant indépendant de ce langage particulier[2].

[1] Évolutions par rapport à la version de janvier 2016 (liées pour l'essentiel aux retours d'expériences suite aux enseignements aux professeurs des écoles stagiaires des années 2015-2016 et 2016-2017, aux exposés devant différents publics et aux commentaires de lecteurs) : refonte du plan général ; affinement des différents types d'objectifs pédagogiques que l'on peut considérer ; réorganisation, affinement et compléments des exemples proposés ; section spécifique sur la différence entre « approche algorithmique » et « approche centrée créativité » ; différents compléments et détails.

Enseigner la pensée informatique avec Scratch amène à mettre au cœur de la réflexion l'algorithmique et la programmation, dans une approche traditionnelle ou orientée « informatique créative ». C'est sensiblement différent que d'enseigner l'« informatique », i.e., la discipline informatique, qui amène à considérer d'autres notions (e.g., de machine, d'automate ou de langage). Je fais ce choix car, à mon sens, c'est bien la « pensée informatique » (on pourrait dire également « pensée algorithmique ») qui doit être le cœur de l'enseignement à l'école élémentaire, ce qui n'empêche pas bien sûr d'aborder, au passage, d'autres aspects.

Le texte est principalement écrit pour des professeurs des écoles (instituteurs et institutrices)[3]. Il s'adresse donc à des lecteurs qui, pour la plupart, n'ont pas suivi de formation à l'informatique[4] 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 maîtrise de ce que l'on veut enseigner (comme on le verra, c'est abordable quelle que soit sa discipline d'origine, et ce document est 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). Par souci de simplification pédagogique, je prends par moment un peu de liberté avec les canons de la discipline (quelques approximations qui n'induiront pas de mauvaises conceptions je l'espère).

Étant 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 aborde successivement l'enseignement de l'informatique de façon générale, puis l'enseignement de la pensée informatique, puis les ressources utilisables. Il est structuré autour de la liste de questions suivantes :

1. Pourquoi enseigner l'informatique à l'école primaire ? (arguments proposés ici et là, programmes actuels, confusions et mauvaises interprétations à éviter, position personnelle).
2. Que faut-il faire pour enseigner l'informatique à l'école ? (il n'y a pas besoin d'être ou de devenir « informaticien », c'est accessible à tous les professeurs des écoles, mais il faut avoir les idées claires sur ses objectifs pédagogiques).
3. Quels objectifs pédagogiques peut-on considérer ? (il est important de dissocier des objectifs qui peuvent être de natures très différentes).
4. Qu'est-ce que c'est que la « pensée informatique » et pourquoi l'enseigner ? (définitions, intérêt, connaissances requises, etc.).
5. Que veut dire « utiliser Scratch pour enseigner la pensée informatique » ? (principes généraux, puis une série d'exemples thématiques : un exercice faisant écrire aux élèves un algorithme ; un exercice de « pensée informatique », illustrant par ailleurs la différence entre « utiliser un programme » et « construire un algorithme/programme » ; un exercice couplant maths et pensée informatique, illustrant par ailleurs comment aligner le moyen « Scratch » avec l'analyse pédagogique/didactique ; un exercice couplant production d'écrit et pensée informatique ; et enfin un exercice autour de la résolution de problèmes).

[2] 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.

[3] Ce document sert de support à un enseignement à l'Espé (École supérieure du professorat et de l'éducation) de Grenoble.

[4] 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.

6. Comment aborder l'enseignement de la pensée informatique ? (il y a deux approches non-disjointes mais significativement différentes, l'approche « orientée algorithmique » et l'approche « orientée créativité »).
7. Que faut-il comprendre à Scratch en tant qu'enseignant, et comment s'y prendre ?
8. Comment définir et gérer des situations pédagogiques ?
9. Quelles difficultés peut-on attendre/anticiper ?

Le texte a une logique linaire. Les sections qui abordent le cœur du sujet sont les sections 4 et 5. Il est possible de se faire une idée intuitive de ce que signifie « initier les élèves à la pensée informatique et à la programmation avec Scratch » en lisant directement la Section 5. Si l'on ne connaît pas du tout la notion d'algorithme ou Scratch, il est possible de jeter un œil sur la Section 4.2 avant de lire le reste du document. Bien que les principes abordés ne soient pas liés à un langage de programmation particulier, les exemples proposés sont tous en Scratch.

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. J'enseigne également à l'Espé de Grenoble (ce cours, et une initiation à la recherche en éducation). 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[5]. 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[6].

1. 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, les confusions à éviter, puis mon point de vue personnel.

1.1. 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[7]. De façon synthétique :

1. 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 ».

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

[6] 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), ainsi que des échanges avec les étudiants, enseignants, collègues, inspecteurs, conseillers pédagogiques (etc.) devant qui j'ai présenté certains éléments de son contenu. Ce qui n'engage pas leur responsabilité dans le texte final bien sûr.

[7] 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

2. 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[8] (auteurs proches du précédent document). De façon synthétique :

1. Arguments clés : l'informatique est une discipline scientifique clé ; nécessité de comprendre le monde et « illettrisme informatique » actuel ; retard de la France ; emplois.
2. 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 ».

Comme on le voit, la focalisation est ici sur la discipline informatique et ses concepts fondamentaux, ce qui inclut l'algorithmique mais également d'autres aspects et compétences.

1.2. Les programmes actuels

Dans le « socle commun de connaissances, de compétences et de culture » on trouve : « [L'élève] sait que des langages informatiques sont utilisés pour programmer des outils numériques et réaliser des traitements automatiques 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. »[9].

Dans les programmes officiels actuels de Cycle 3 on peut trouver les choses suivantes[10] :

1. 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 ».
2. 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

[8] 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

[9] Extrait du Bulletin officiel n° 17 du 23 avril 2015. Pour mémoire, « 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 ». Chaque fois que je présente ce point je demande qui, dans la salle, maîtrise ces compétences, ce qui provoque pas mal (une très large majorité) de sourires embarrassés.

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

(…) constituent des moments privilégiés pour une première initiation à la programmation notamment à travers la programmation de déplacements ou de construction de figures ».

3. Matériaux et objets techniques / attendus de fin de cycle / « Les élèves apprennent à connaître 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 maîtrisent 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 »[11].

Sur Eduscol[12] on trouve notamment des choses dans les sections : « Mathématiques / Espace et géométrie / Initiation à la programmation aux Cycles 2 et 3[13] (exemples et indication pédagogiques autour de la géométrie) ; « Sciences et technologie / Mettre en œuvre son enseignement / Matériaux et objets techniques » (activités autour de robots qui incluent les intentions pédagogiques « découvrir et approfondir la notion d'algorithme ; apprendre à coder ») ; et également des ressources de formation des enseignants sur des aspects très techniques comme le « stockage des données ».

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 et/ou la pensée informatique, il faut faire tout particulièrement attention à :

1. Ne pas confondre 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 (cf. maintenant le « Cadre de référence des compétences numériques »).
2. Ne pas confondre avec 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.

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. 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

[11] 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 ; 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.

[12] Site du ministère de l'éducation nationale qui propose « un ensemble de sites et de services dédiés pour informer et accompagner les professionnels de l'éducation »; http://eduscol.education.fr/pid34150/cycle-3.html

[13] http://cache.media.eduscol.education.fr/file/Initiation_a_la_programmation/92/6/RA16_C2_C3_MATH_inititation_programmation_doc_maitre_624926.pdf

tout à fait possible d'enseigner l'informatique sans rien connaître de la bureautique, des TBIs ou des logiciels de géométrie dynamique (il faut des compétences, mais pas celles-là).

Plus généralement, l'enseignement de l'informatique et de ses compétences sous-jacentes ne sont pas intrinsèquement liées à l'utilisation d'ordinateurs. En particulier, on peut faire de l'algorithmique sans ordinateur (informatique « débranchée » ou « unplugged »)[14]. Le lien entre algorithmique et ordinateur est simplement qu'un ordinateur permet d'exécuter (de « faire tourner ») des algorithmes une fois transformés en programmes, i.e., écrits dans un langage de programmation interprétable par l'ordinateur. 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 (et qui me semble très pertinente à l'école), mais ce n'est pas la seule possible. Certains acteurs (cf. notamment « La main à la pâte ») donnent d'ailleurs une grande importance à l'approche « débranchée ».

1.4. Point de vue personnel

A la question « Que faut-il enseigner à l'école primaire et comment » ma réponse est : les compétences sous-jacentes à ce que l'on appelle la « pensée informatique », i.e., les aspects liés à la résolution de problèmes, l'algorithmique, la logique, l'abstraction ; en abordant ces compétences à travers des exercices ou projets impliquant d'autres disciplines ; et, éventuellement, en en profitant pour initier les élèves à d'autres aspects (comment fonctionne un ordinateur, etc.) quand une opportunité se présente. Je développe ce point en Section 4.

A la question « Faut-il enseigner l'informatique à l'école primaire ? », ma réponse est : « Je n'en sais rien ». Je sais que c'est faisable. Je pense que c'est pertinent. Mais pour *savoir* si c'est pertinent et pourquoi, il faut mener tout un ensemble de travaux scientifiques : 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 cela permet, à travers des exercices pluridisciplinaires, d'enseigner d'autres matières de façon pertinente (e.g., plus efficace pour certains élèves ?) est-ce qu'il y a transfert vers d'autres domaines ? etc. Mener des actions exploratoires en classe et/ou des expérimentations scientifiques[15] permettront de répondre à ces interrogations (par le retour des enseignants, par les analyses scientifiques qui pourront être faites).

2. Que faut-il faire pour enseigner l'informatique à l'école ?

Pour enseigner l'informatique à l'école il faut, comme pour tout enseignement :

1. Disposer ou construire une certaine conceptualisation du domaine considéré et des enjeux d'apprentissage (identifier, définir, dissocier, les notions/compétences en jeu).
2. Identifier les objectifs pédagogiques que l'on se propose de considérer.
3. Maîtriser les notions/compétences en jeu et les moyens de les faire travailler par les élèves (types d'exercices, etc.). Ce qui inclut, si l'on se propose de faire travailler les élèves sur la construction de programmes, 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).

[14] 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 enchaînant une séquence d'actions comme « avancer de 1 pas », « tourner droite » et « tourner gauche ». 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).

[15] Cf. par exemple le projet EXPIRE (www.caissedesdepots.fr/sites/default/files/medias/projet_9_dp_cdc.pdf).

L'écueil majeur est le point « disposer/construire une certaine conceptualisation du domaine considéré ». Pas par sa difficulté, mais par l'inquiétude qu'il engendre.

Dans mon expérience personnelle d'enseignement (en formation initiale et continue), l'immense majorité des professeurs des écoles ont une vision très confuse de l'informatique. La raison en est simple : c'est une discipline qu'ils n'ont pour la plupart jamais abordée en tant que telle. Par suite, ils en ont une perception par défaut qui est celle de l'utilisateur (d'un ordinateur, de logiciels de bureautique, etc.). Ceci ne permet pas de comprendre ce qu'est la discipline informatique ou la « pensée informatique » et, pire, crée des confusions. L'informatique, c'est souvent un domaine mal défini mêlant « savoir utiliser un ordinateur » et « savoir comment marche un ordinateur », et dont on se méfie car on n'y connaît/comprend pas grand-chose, d'autant que « souvent, ça ne marche pas ». Quant à « l'informatique à l'école », c'est faire utiliser aux élèves un programme existant (e.g., un jeu à vocation éducative) ou un outil de bureautique. La notion de « pensée informatique » et même souvent d'algorithme est inconnue. Ce qui tombe bien car si, de façon générale, « faire de l'informatique » génère une certaine appréhension, alors s'il s'agit de construire des programmes et pas simplement de les utiliser !

Les enseignants ne sont bien évidemment en rien responsables de cet état de fait, c'est simplement une conséquence de leur formation.

Cette vision confuse amène, légitimement, à penser que non seulement on ne maîtrise pas mais, également, qu'il sera difficile/impossible de maîtriser l'objet d'enseignement « informatique », tant disciplinairement que didactiquement. Et donc à s'inquiéter à l'idée de devoir l'enseigner.

Si l'on aborde les choses en termes généraux, effectivement, il y a de quoi s'inquiéter. Tout d'abord parce que l'informatique, en tant que domaine scientifique et technique, est un domaine complexe, 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 également 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 enfin 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). 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 enseigner, de légitimité disciplinaire, de création de postes, etc. Enfin, et c'est le principal problème, le travail qui consiste à aider les enseignants (études didactiques, réunions de consensus, retours d'expériences, etc.) est actuellement très peu avancé, et une large part reste donc à la charge de l'enseignant.

Tout cela n'aide pas à aborder sereinement l'enseignement de l'informatique.

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 de mathématiques 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.). Pourtant, ceci n'inquiète pas outre mesure les enseignants.

On touche ici un point qui différencie l'informatique des maths, du français, de l'histoire ou de toutes les autres disciplines. La plupart des enseignants actuels n'ont pas rencontré l'informatique dans leur formation d'étudiant mais, contrairement aux autres disciplines, ils n'ont pas non plus rencontré l'informatique comme écolier/écolière, i.e., au niveau où ils vont l'enseigner. Bien sûr, l'enseignement des maths ou de l'histoire ont changé depuis ses années d'écolier/écolière « mais bon, pour ces matières, je sais qu'au niveau CM1-CM2 j'y arriverai ».

Pour enseigner l'informatique à l'école, 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, littéraires, historiens, ou spécialistes des différentes matières

enseignées à l'école. Comme pour les autres matières, le fait de ne pas avoir suivi de cours d'informatique dans son cursus universitaire ne pose pas de problème insurmontable. 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 qui n'aident pas à son enseignement.

Cependant, et c'est le point clé, pour identifier et travailler ce que l'on a besoin de savoir, il faut avoir des idées claires sur ce que l'on veut enseigner (et pourquoi). Ceci permet ensuite de faire la part des choses (ce qu'il faut comprendre, ce qu'il faut savoir utiliser, ce qui est hors sujet). Une fois identifiées les connaissances disciplinaires objet de l'enseignement, on se rend compte qu'elles ne sont pas plus compliquées que celles d'autres matières (maths, français, histoire, etc.) au même niveau d'enseignement[16].

La section suivante essaie de proposer des éléments pour aider à conceptualiser le domaine et identifier des objectifs pédagogiques possibles.

3. Quels objectifs pédagogiques peut-on considérer ?

Le tableau ci-dessous liste quelques objectifs pédagogiques potentiels de l'enseignement de l'informatique (au sens large).

L'idée est, en différenciant différents objectifs possibles, d'aider à conceptualiser un peu le domaine (du point de vue de son enseignement).

objectifs pédagogiques autour de la technologie	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. faire comprendre comment fonctionnent un ordinateur, un réseau d'ordinateurs, etc.	*ce type d'objectif peut être abordé de différentes façons : comme un moyen de démythifier, 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, triptyque données – traitements - résultats, représentation binaire, stockage d'information, processus de calcul, aspects matériels, etc.)*
objectifs pédagogiques autour de l'algorithmique et de la résolution de problèmes	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 « faisant tourner » des algorithmes ; il est possible de travailler sur des algorithmes « de tous les jours » (recette de cuisine, multiplication d'entiers, accord du participe passé, organisation des actions pour se rendre d'une ville à une autre, etc.) et/ou des choses plus spécifiques (tri, recherche dans une liste triée, etc.)*
	faire pratiquer la construction d'algorithmes	*c'est une compétence différente : comprendre un algorithme (comprendre une solution) et inventer/construire un algorithme (trouver une solution) sont deux choses très différentes*
	faire comprendre et pratiquer la démarche idée générale - modélisation - décomposition en sous-problèmes/équipes - algorithme - codage	*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*

[16] A la question « Je savais ce qu'était la pensée informatique et la programmation avec Scratch avant le cours », les réponses des 142 étudiants (professeurs des écoles stagiaires) qui ont suivi cet enseignement à ce jour sont : oui (3) ; un peu (6) ; non (133). A l'issue du cours, 59 des professeurs des écoles stagiaires indiquaient « si j'en ai l'occasion, je me lance » et 76 de plus « si j'en ai l'occasion, je me lancerai, mais quand j'aurai un peu plus d'expérience dans le métier ». Bien évidemment, il s'agit ici de l'enseignement de l'algorithmique/programmation, à l'école élémentaire, 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.

objectifs pédagogiques autour de la notion de langage	faire comprendre la notion de langage (langage naturel, langage informatique) en tant que telle (comme un système de codage, comme un outil de communication, 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 (?)*
	faire comprendre la notion de langage de 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.)*
	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) ; mais inversement (et que les informaticiens me pardonnent), il est également possible de chercher à faire en sorte que les élèvent sachent programmer leurs algorithmes pour pouvoir les tester, sans vraiment chercher explicitement à ce qu'ils comprennent la notion de langage de programmation en tant que telle*
objectifs pédagogiques autres que informatiques	travailler une compétence non-informatique (maths, français, langue_2, etc.)	*la dimension informatique peut être un simple habillage, pour créer une dimension ludique, motivante, etc. (sans enjeu d'apprentissage du côté informatique) ; par exemple, faire travailler la production d'écrits en créant une scène où différents personnages discutent* *la dimension informatique peut être introduite pour obliger à travailler une compétence cible ; par exemple, faire travailler une compétence en logique (conjonction, disjonction, négation), en maths, en histoire (etc.) en travaillant sur un exercice où une expression logique (ou le calcul de maths, ou une information historique) est nécessaire à l'écriture de l'algorithme* *enfin, ce peut être un exercice pluridisciplinaire, mêlant objectifs pédagogiques en informatique et dans d'autres matières*
objectifs pédagogiques autour de la créativité	faire en sorte que les élèves développent leur créativité	*c'est un objectif phare dans la vision US et, du coup, présente dans beaucoup de manuels/ressources ; cf. Section 6*
	faire en sorte que les élèves sachent exprimer leurs idées créatives (jeux, dialogues, fictions, etc.) en termes de problèmes à résoudre, d'algorithmes et de programmes	*cela rejoint les objectifs autour de la résolution de problèmes et de l'algorithmique, 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 l'on choisit ses phrases (ce qui permet au passage d'éviter celles où l'on ne sait pas accorder le verbe) ; mais les choses se rejoignent à un certain niveau bien sûr ; cf. Section 6*

objectifs pédagogiques autour de dimensions relatives à l'éthique, la nature des informations et processus du monde numérique, etc.	faire en sorte que les élèves réfléchissent à des questions de propriété intellectuelle, de traces numériques, de qualité (véracité, précision, etc.) de ce que calcule un programme, etc.	*il s'agit d'aborder des questions d'ordre général (éducation civique, philosophie, etc.)*
contre-exemples : objectifs ou activités qui ne relèvent pas d'un enseignement de l'informatique	faire pratiquer l'utilisation d'un ordinateur et/ou des logiciels de bureautique (éditeur de texte, etc.)	*c'est un objectif tout à fait légitime, mais qui ne relève pas de l'enseignement de l'informatique ; c'est l'objet de formation type B2i, et clairement pas l'objet abordé dans ce document*
	faire utiliser un programme (créé avec Scratch ou pas) par des élèves	*l'utilisation d'un programme tout fait n'amène en général pas à travailler des compétences informatiques ; cf l'exemple des fractions en Section 5.3*

Cette liste est à prendre comme un support de réflexion. Elle n'est sans doute pas complète, et pourrait être structurée autrement. Elle illustre cependant qu'il est possible de construire des séquences pédagogiques abordant « de l'informatique » avec des objectifs pédagogiques très différents ; que ces objectifs peuvent n'être ni exclusifs ni disjoints ; que certains objectifs sont (aussi) des moyens pour atteindre d'autres objectifs. Elle illustre également que l'enseignement de l'informatique peut 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.

Identifier et contraster les objectifs pédagogiques que l'on peut/veut considérer est une condition sine qua non de tout enseignement. En informatique (mais c'est sans doute également le cas dans bien d'autres matières), la tâche est complexe car les objectifs ne sont pas disjoints, et se recoupent de différentes façons. C'est justement pour cela que cet effort doit être fait avec attention. Ainsi qu'indiqué précédemment, la conceptualisation proposée ci-dessus est *une* conceptualisation, qui peut être discutée / modifiée / complétée. Ce qui est important, c'est de savoir ce que l'on veut enseigner (et pourquoi). « Faire de l'informatique en classe », ou même « faire du Scratch », cela peut correspondre à des réalités très différentes, et donc cela ne veut rien dire.

4. Qu'est-ce que c'est que la « pensée informatique » et pourquoi l'enseigner ?

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

Le terme « pensée informatique » a fait florès depuis la parution d'un article « point de vue » d'une chercheuse en informatique aux États Unis[17] 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[18], 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. » En France un article souvent cité[19] présente les choses ainsi : « [Le] souci du caractère algorithmique de la description des objets, du langage dans lequel ces descriptions sont

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

[18] 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.

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 la prise en compte de l'enseignement de l'informatique dans les programmes du primaire et du secondaire, cf. Section 1.

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[20] :

1. « [La pensée informatique] implique d'appréhender le monde selon l'approche employée en programmation par les développeurs de logiciels.
2. 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) ».

Pour ma part, la façon dont je présente habituellement les compétences sous-jacentes à la pensée informatique est : savoir décomposer un problème en sous-problèmes plus simples ; savoir réfléchir aux tâches à accomplir pour résoudre un problème en termes d'étapes et d'actions (algorithme) ; savoir décrire les problèmes et les solutions à différents niveaux d'abstraction, ce qui permet d'identifier des similitudes entre problèmes et, par suite, de pouvoir réutiliser des éléments de solutions.

La pensée informatique ne se réduit donc pas à l'algorithmique, mais le concept d'algorithme est au cœur de la pensée informatique.

4.2. Algorithmique : de quoi parle-t-on exactement ?

Un algorithme est un enchaînement 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.

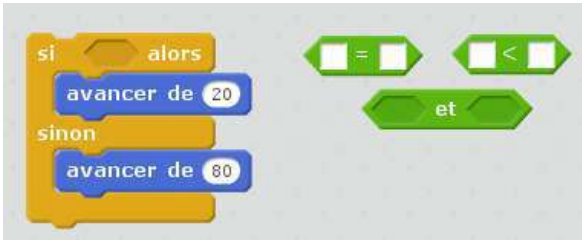
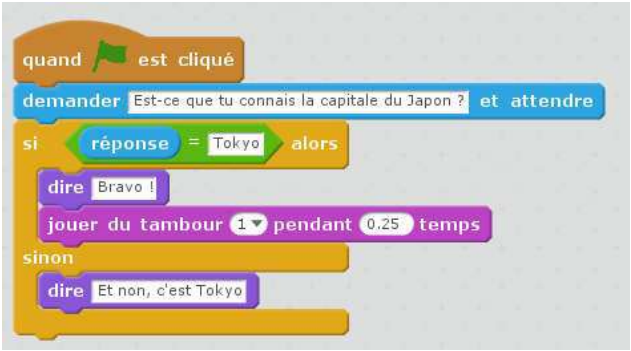
La notion d'algorithme n'est pas propre à l'informatique : le programme de Cycle 2 de maths indique la compétence cible « Mettre en œuvre un algorithme de calcul posé pour l'addition, la soustraction, la multiplication ». Une recette de cuisine est un algorithme, dont les actions sont de « casser les œufs », de « mettre de la farine » ou de, si nécessaire, « rajouter un peu de sel ».

Les notions à partir desquelles on construit des algorithmes sont peu nombreuses, et on va pour l'essentiel retrouver les mêmes à l'école, collège, lycée et Université (à des degrés de complexité différents bien sûr). A mon sens, il est possible d'aborder en CM1-CM2, voire en CE, les notions/compétences présentées ci-dessous (avec leur représentation dans le langage de programmation Scratch)[21] .

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

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

[21] La question de savoir s'il faut, à l'école élémentaire, dissocier les notions d'algorithme et de programme (traduction de l'algorithme dans un langage de programmation, par exemple Scratch) est, à mon sens, une question ouverte. En ce qui me concerne, au niveau du Cycle 3, je n'en vois pas l'utilité (écrire un algorithme ou écrire un programme avec Scratch, c'est quasiment la même chose). 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.

Notion d'algorithme / de programme. En Scratch un programme (ou « script ») est une série d'actions associées à un personnage[22] 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.	
Notion d'action / instruction : « dire », « avancer », « jouer un son », mettre le stylo en position « écriture » (quand le personnage va se déplacer il va laisser une trace), etc. Certaines actions vont utiliser des opérateurs (par exemple, « + »). Séquence d'actions = lier plusieurs actions (en rapprochant les blocs, qui vont s'encastrer façon puzzle), par exemple « dire bonjour » puis « avancer ».	
Structure conditionnelle (Si-Alors-Sinon) : faire en sorte que certaines actions soient effectuées en fonction d'une condition (ce qui mobilise donc la logique et des opérateurs comme « = » ou « < »).	
Boucle (ou « structure itérative ») : faire répéter un ensemble d'actions un certain nombre de fois. Scratch propose 3 variantes.	
Éventuellement, utilisation de variables[23] prédéfinies dans le langage (e.g., en Scratch, suite à l'action « demander », ce que tape l'utilisateur est stocké dans une variable prédéfinie « réponse ») ou de variables prédéfinies par le concepteur de l'exercice ; dit autrement : faire utiliser des variables à l'élève, mais sans les lui faire définir lui-même (cf. exemples illustratifs en Section 5). A noter : la séquence d'actions permettant de demander et récupérer des informations via une question à l'utilisateur aura souvent la même structure (question + saisie réponse) ; elle peut être vue comme une solution abstraite réutilisable dans d'autres programmes.	

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

[23] 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 va 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 « age ».

Éventuellement (question ouverte) : proposer des exercices où les élèves doivent définir eux-mêmes des variables[24].	

Selon le temps consacré, il est bien sûr possible d'aborder d'autres choses, en particulier les mécanismes de synchronisation : via des délais (attendre x secondes puis faire telle action) et/ou via des envois de messages (aborder ce type de technique en tant que telle me semble compliqué/inutile, mais ce peut être introduit comme un moyen de créer des animations ; cf. exemples illustratifs en Section 5).

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 »).

4.3. Pourquoi enseigner la pensée informatique ?

L'enseignement de la pensée informatique relève fondamentalement des objectifs pédagogiques autour de l'algorithmique et de la résolution de problèmes indiqués en Section 3.

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 :

1. Les compétences sous-jacentes à la pensée informatique (l'algorithmique, la logique, la décomposition de problèmes en sous-problèmes, les processus d'abstraction, de généralisation et d'instanciation) sont des compétences utiles et importantes à tous et pas simplement aux informaticiens, que l'on retrouve dans la vie de tous les jours (organisation de l'action).
2. Il est possible d'enseigner ces compétences de différentes façons, y compris sans informatique. L'approche algorithmique/programmation me semble cependant présenter des propriétés intéressantes. Elle amène notamment à travailler sur des réifications. Par exemple, quand on travaille sur un algorithme et qu'on veut le rendre « plus abstrait », on travaille sur un objet concret, qu'on voit à l'écran et qu'on modifie. 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).
3. La pratique de l'algorithmique 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 la pensée informatique et d'autres domaines. Il est également possible de coupler enseignement de la pensée informatique et créativité (cf. Section 6). Cela ouvre un large champ pour l'exploration d'innovations pédagogiques tissant des liens entre différents apprentissages (exercices ou de

[24] Mon expérience est que les élèves utilisent très facilement (très intuitivement) des variables prédéfinies pourvu qu'elles aient un nom bien choisi. Mais ceci ne signifie pas qu'ils maîtrisent parfaitement la notion de variable. Les avis divergent sur la capacité (ou la facilité avec laquelle) des élèves peuvent maîtriser la notion de variable et s'en servir de façon pertinente. 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. La durée de l'enseignement est bien sûr un facteur important.

projets multi-objectifs et pluridisciplinaires)[25]. A ce niveau, le contexte de l'école élémentaire, où un même enseignant organise et conduit un enseignement polyvalent, semble particulièrement propice.

4. Les points d'entrées « construction d'algorithmes » et « programmation » sont des moyens que certains élèves peuvent percevoir (ou percevoir à certains moments) comme ludiques et motivants (notamment par la dimension de défi de ce type d'activité).
5. Les points d'entrées « construction d'algorithmes » et « programmation » permettent, en passant, d'aborder des objectifs de plus hauts niveaux comme : démythifier (un ordinateur, ce n'est pas magique, ce qu'il fait est régi par 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.

Se focaliser sur la pensée informatique est sensiblement différent du fait de considérer la *discipline informatique* (typiquement, certains « concepts fondamentaux » ne sont pas abordés en tant que tels, de même que les aspects techniques de comment fonctionne un ordinateur ou comment on code/transmet des informations). Ces objectifs sont bien évidemment parfaitement légitimes, mais je ne les mettrai pas au premier plan à l'école élémentaire (ils seront abordés explicitement au Cycle 4)[26].

A noter : comme on l'a vu en Section 4.2, enseigner la « pensée informatique » à l'école primaire (et connaître les instructions Scratch nécessaires) ne nécessite qu'un investissement disciplinaire et technique limité et, en aucun cas, de « devenir un informaticien ». Encore une fois : il s'agit de savoir faire des choses … que l'on va demander à des élèves de CM1-CM2 de savoir faire !

En particulier, pour ce qui est de Scratch, il est tout particulièrement important de comprendre le point suivant. L'objet d'enseignement, c'est la pensée informatique (comment on s'y prend pour analyser le problème, écrire une solution en terme d'algorithmes, etc.) ; ce n'est pas le langage de programmation en tant que tel, qui n'est qu'un *moyen* de mettre en œuvre l'algorithme sur la machine. Scratch peut donc être abordé et traité comme une ressource, i.e., l'enseignant doit savoir comment l'utiliser pour résoudre les exercices qu'il propose, mais n'a pas à maîtriser tous ses aspects (ce qui n'est pas le cas pour un objet d'enseignement bien sûr). Et, s'il y a une question inattendue (« comment on fait pour faire grossir le personnage ? »), et bien, … on cherche ensemble comment faire (cf. Section 7).

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 faisant pratiquer les compétences de la pensée informatique en utilisant Scratch comme support ». Après des définitions permettant de dresser un cadre commun, je présente différents exemples[27] qui, en plus d'être illustratifs, abordent des points particuliers.

[25] 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.

[26] Rien n'empêche de les aborder incidemment bien sûr. Une remarque complémentaire. L'étude d'objets informatiques (une page Web, une image numérique, etc.) ou de logiciels d'usages courant (un logiciel de messagerie, un navigateur Web, etc.) est souvent utilisée comme prétextes pour aborder l'algorithmique. Par exemple, la façon dont une information (texte ou image) est codée dans la machine peut permettre d'introduire des algorithmes de codage, et donc faire travailler l'algorithmique. Si l'on veut enseigner la notion d'information et la notion de codage et l'algorithmique, cela semble idéal. Mais si la focalisation est sur la pensée informatique, des supports plus proches des matières compétences pratiquées par ailleurs par les élèves (maths, français, créativité, etc.) me semblent plus pertinents.

[27] Les exemples illustrent par ailleurs les notions d'un « programme possible » évoquées en Section 4.2, y compris celles-qui relèvent d'une utilisation avancée (variables, envois de messages). Ils sont accessibles en ligne à : https://scratch.mit.edu/studios/1808925/.

5.1. Définitions et principes généraux

Dans cette section, je vais m'appuyer sur les définitions suivantes[28] :

1. Une situation pédagogique est une situation conçue pour amener un élève à développer une activité favorable à l'atteinte d'un ou plusieurs objectifs pédagogiques.
2. Concevoir et gérer une situation pédagogique consiste généralement à :
   - ✗ 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)[29]. 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 d'exercice faisant écrire aux élèves un algorithme

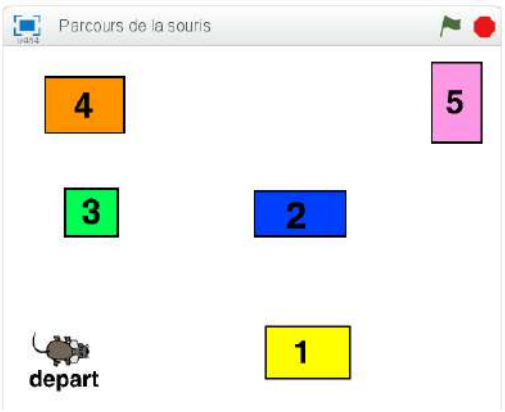
L'exercice (cher à la pédagogie Logo) le plus simple et le plus classique pour faire comprendre ce que c'est que l'écriture d'un algorithme est probablement le pilotage d'un objet mobile (robot, personnage, etc.). Il est en effet très intuitif : on comprend bien que pour faire bouger un personnage il faut qu'il effectue des actions (avancer, tourner à droite, etc.)[30].

L'exemple ci-dessous illustre ce type d'exercice (attention, il « cache » une idée sur laquelle nous allons revenir en section 5.4 : aligner l'analyse pédagogique avec l'outil Scratch, en créant des blocs spécifiques).

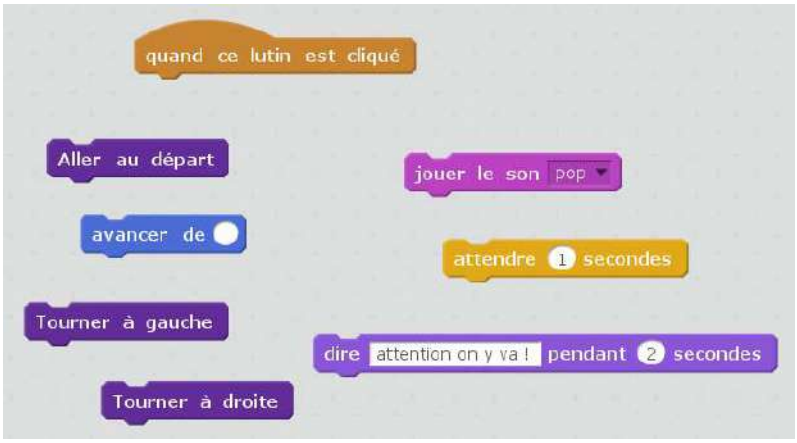
[28] Reprises et simplifiées de : « Précis de recherche en ingénierie des EIAH » (P. Tchounikine, 2009 ; en ligne à http://lig-membres.imag.fr/tchounikine/).

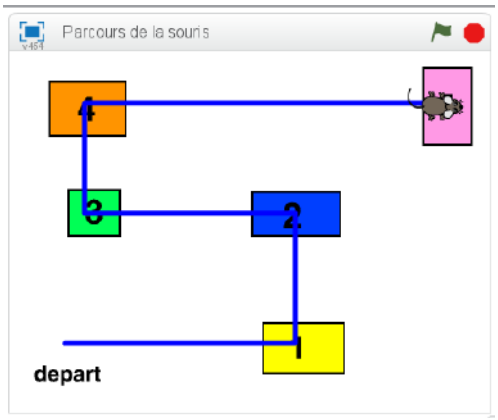
[29] Pendant la séance il peut s'avérer nécessaire de modifier (réduire, simplifier, complexifier, étendre) la tâche-élève prévue, il peut s'avérer pertinent de changer la tâche pour saisir une opportunité, etc.

[30] C'est le type d'exercice que propose notamment Eduscol, en indiquant « Aux Cycles 2 et 3, les ambitions sont assez modestes : il s'agit de savoir coder ou décoder pour prévoir ou représenter des déplacements, de programmer les déplacements d'un robot ou ceux d'un personnage sur un écran. Des activités géométriques, consistant en la construction de figures simples ou de figures composées de figures simples, sont également proposées. (…) il s'agit d'anticiper un déplacement complet, de prévoir à l'avance l'ensemble des instructions permettant d'obtenir un déplacement complexe souhaité ».

<table>
<tr>
<td> 
</td>
<td>La scène : un personnage (la souris) et un ensemble de repères en fond de scène.

La consigne : construire l'algorithme/programme qui permet à la souris de passer par les 5 blocs.

Compétences mises en jeu : orientation, mesure et écriture d'algorithme.</td>
</tr>
<tr>
<td> 
</td>
<td>La zone où les élèves doivent construire l'algorithme/programme. Ici, pour faire de l'étayage, les blocs utiles sont proposés (Aller au départ, avancer, Tourner à droite, Tourner à gauche), ainsi que des blocs pour « faire plus fun » (jouer un son quand on passe sur un repère, écrire un message, etc.).</td>
</tr>
<tr>
<td> 


</td>
<td>Une solution (un algorithme/programme, avec les blocs utiles uniquement) et le résultat de l'exécution du programme.</td>
</tr>
</table>

Cet exercice a été testé en classe pleine (29 élèves, 11 ordinateurs), après une explication de 3 minutes (ce que fait un bloc « avancer », comme lier les blocs entre eux), à des élèves qui, à l'exception de 3, n'avaient jamais utilisé Scratch. Ils ont mis entre 5 et 10 minutes pour le résoudre.

Sur ce canevas, il est possible de construire une variété d'exercices (dessiner un carré, dessiner une maison, etc. ; cf. les ressources Logo[31]). Une remarque en passant : ce type d'exercice peut également être fait en activité « débranchée », dans la cour par exemple, un élève jouant le rôle de la souris.

Cet exemple permet de comprendre ce qu'est un algorithme (un enchaînement d'actions). Cependant, l'envers de son côté très intuitif est qu'il est également assez peu représentatif de la variété des problèmes que l'on peut aborder. A mon sens, c'est un exemple utile mais dont il faut se méfier (il peut amener à des représentations peu pertinentes).

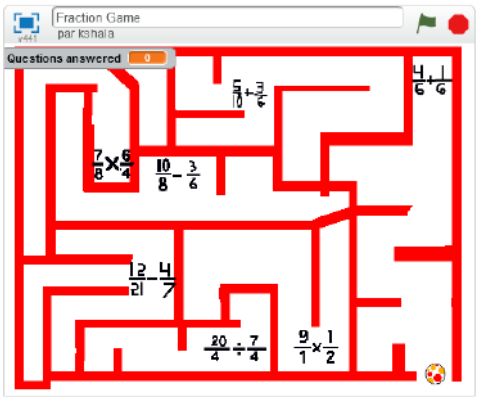
[31]cf . par exemple http://www-irem.univ-paris13.fr/site_spip/spip.php?article830.

5.3. Un exemple d'exercice de « pensée informatique » (et illustrant la différence entre « faire utiliser un programme » et « faire construire un algorithme/programme »)

Voici trois 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).



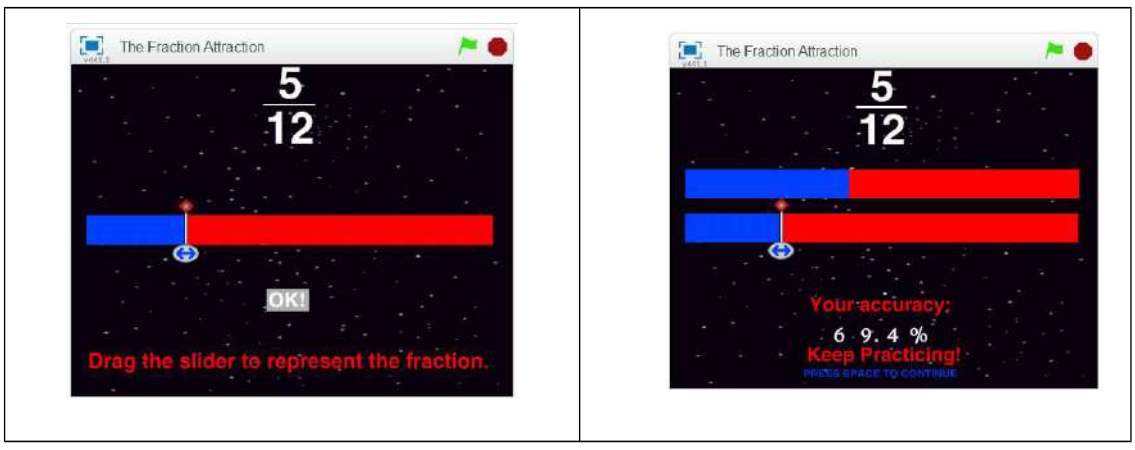
Ici, Scratch est utilisé pour créer un contexte ludique. L'élève-utilisateur ne fait pas d'algorithmique ou de programmation : il utilise un programme tout prêt. Le système apporte une rétroaction de base : il indique si la réponse est correcte ou pas en ouvrant les portes du labyrinthe ou pas.

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 programme met ensuite 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, ainsi qu'une rétroaction moins binaire que dans l'exemple précédent (solution correcte et % de réussite).

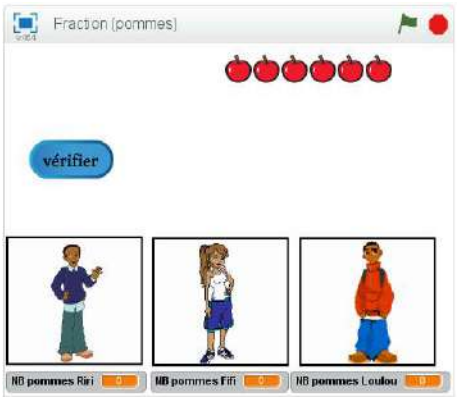
L'activité espérée est : l'élève-utilisateur calcule les fractions et identifie un ordre de grandeur.

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).

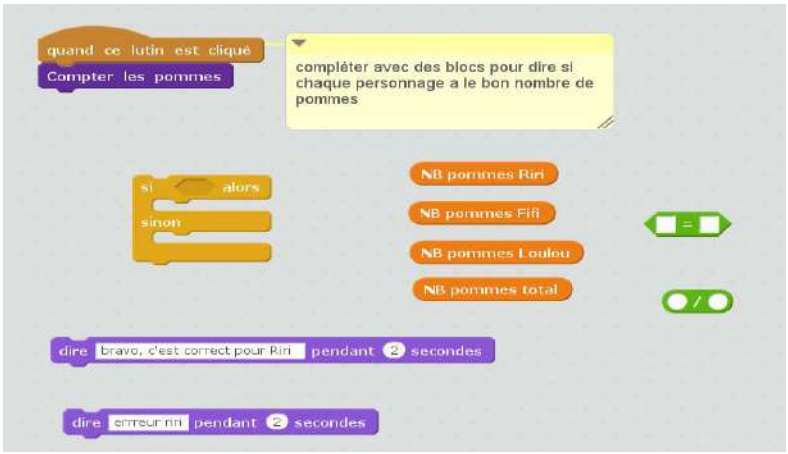
Un exercice d'algorithmique/programmation (que j'ai construit sur la base d'exercices de maths de CM1/CM2)

L'idée générale est de proposer aux élèves de faire un programme qui pose un problème à l'utilisateur et indique si la solution est correcte.

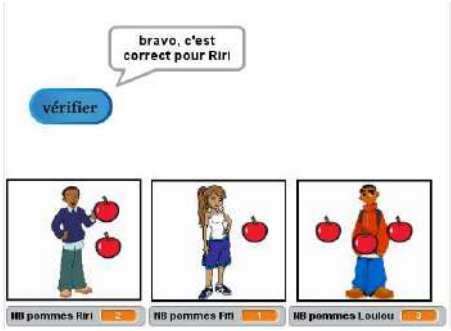
L'architecture générale du programme est prédéfinie : il y a 3 personnages (Riri, Fifi et Loulou), des variables prédéfinies (nombre de pommes total, nombre de pommes de chacun) et des scripts (pas montrés à l'élève) pour calculer le nombre de pommes de chacun (quand on approche une pomme d'un personnage,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

	La scène.

La tâche-élève est de définir le script du bouton « vérifier », i.e., l'algorithme qui vérifie si les pommes sont bien réparties 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-Sinon », des opérateurs de comparaison, et les variables 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[32] » (lorsque l'on va cliquer sur le bouton « vérifier »).

[32] 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.

	Un début de solution pour Riri (le programme complet doit tester également le nombre de pommes de Fifi et Loulou).
	Le résultat en cours d'exécution.

L'activité espérée a ici une double dimension. D'une part, l'élève 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 trivial, ce qui fait que cet exercice, posé au niveau CM, n'est pas un exercice de maths (les élèves savent diviser 6 par 3). D'autre part, l'élève 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 !) mais, surtout, de l'algorithmique (et de la 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).

Pour creuser la dimension « pensée informatique » de l'exercice, j'indique ci-dessous les petits tests de cet exercice avec des élèves de CM2 (situation = 3 élèves et moi comme tuteur, avec 3 groupes différents). Très naturellement, les élèves ont commencé par décomposer le problème en sous-problèmes (« on va faire pour Riri et après pour les autres »). Les élèves 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, reconnaissance de similitudes entre problèmes). Et, dès qu'ils ont vu que l'on pouvait dupliquer le bloc représentant la solution pour Riri en cliquant dessus, ils l'ont dupliqué 2 fois et ont changé les variables (nombre de pommes de Fifi) et les messages affichés (adaptation de la solution d'un problème similaire). Tous ont écrit une condition logique de type « NB pomme Riri = 2 »[33]. La question « pourquoi ? » a amené la réponse « parce qu'il y a 6 pommes et 3 personnages », et la question « pourquoi 6 ? » la réponse « il faut diviser le nombre de pommes total par 3 ». On est alors dans un processus d'abstraction : description d'une solution plus générale (qui marche avec 9 ou 12 pommes), qu'ils ont su écrire immédiatement (utilisation de la variable NB pommes total et de l'opérateur de division). L'étape suivante serait de passer à « en fait il faut diviser le nombre de pommes par le nombre de personnages ». 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.

[33] Dans cet exercice, les élèves se voient proposer des variables prédéfinies (« NB pomme Riri », etc.), qu'ils/elles ont utilisées sans hésitation. Ceci ne veut pas dire qu'ils ont compris la *notion* de variable.

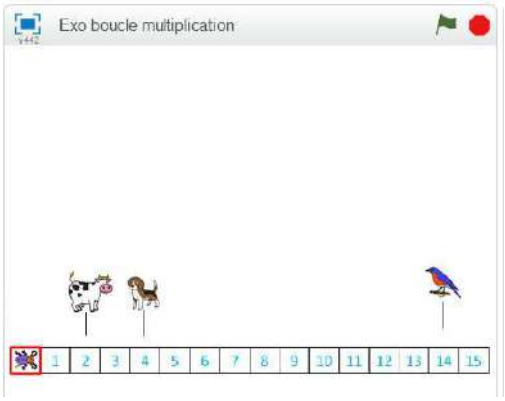
La situation (ici, triviale) permet différentes variantes. Les variantes peuvent permettre de jouer sur différents registres, par exemple :

1. 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.
2. 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.
3. 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.
4. 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[34].

5.4. Un exemple d'exercice couplant maths et pensée informatique (et illustrant l'alignement du moyen « Scratch » avec l'analyse pédagogique/didactique)

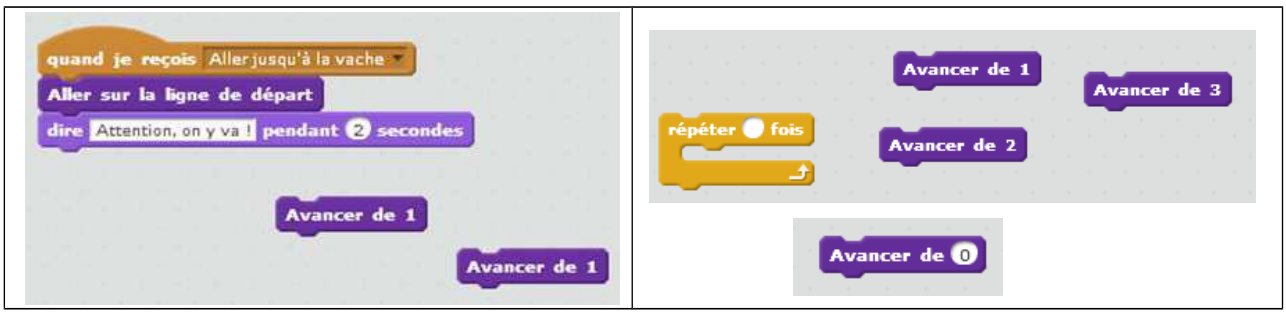
Soit l'objectif d'amener les élèves au concept de multiplication en les faisant travailler sur des multiplications par additions réitérées. Un exemple de mise en œuvre est un exercice où il faut écrire un programme faisant avancer un personnage (ici, un insecte) de plusieurs pas pour aller d'un point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

Pour ce type d'exercice, les travaux en didactique des maths identifient deux variables didactiques[35] : 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 ». C'est 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.

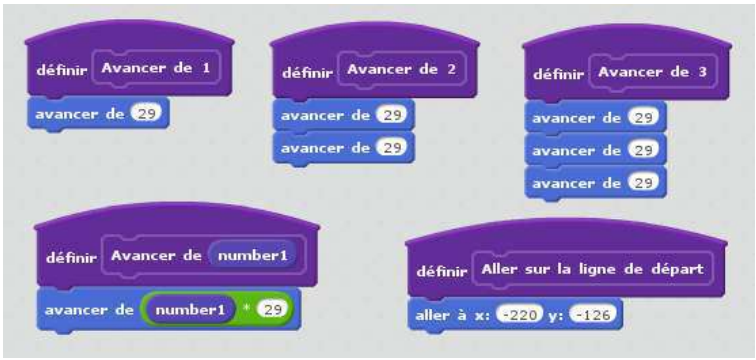
[34] Pour ceux qui veulent pousser plus loin, une autre variante possible est que les données et/ou la consigne soient produites automatiquement (par exemple, en générant des valeurs aléatoires).

[35] 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.



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. L'exercice a été testé en classe pleine de CM2 et réalisé par tous les élèves en ~5-10 minutes (du point de vue des maths, c'est de niveau CE, et l'utilisation du « Répéter » pour aller jusqu'à l'oiseau a été immédiate). Bien évidemment, c'est une trame que l'on pourrait complexifier.

Dans cet exercice, 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 »). Pourquoi ? Parce que faire chercher aux élèves si pour avancer d'une case il faut avancer de 20, 25 ou 30 pixels, c'est sans intérêt pédagogique (étant donnés les objectifs considérés ici). Et que, par ailleurs, le point où se trouve la ligne de départ a des coordonnées négatives.



L'exemple proposé en Section 5.2 relevait du même procédé : les blocs « tourner à droite » et « tourner à gauche » permettent de gommer la notion d'angle, qui n'est pas un objectif dans cet exercice ; le bloc « aller au départ » permet de repositionner la souris, de la mettre dans la bonne direction, et d'effacer le parcours précédent.

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 (c'est au concepteur de l'exercice de faire cela, concepteur qui n'est pas forcément l'enseignant).

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 d'exercice couplant production d'écrits et pensée informatique

Soit l'objectif de faire travailler la compétence « production d'écrits ». 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 » ; complément possible : « exploiter les différentes attitudes des personnages qui sont proposées.

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 « dit ». Le fait d'utiliser des personnages ayant plusieurs « costumes » dénotant différentes émotions peut être un vecteur pour faire développer un dialogue complexe (consigne d'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.

Un exercice plus ambitieux maintenant : réaliser une fiction interactive ou un jeu. Il faut pour cela construire 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 sur les techniques de gestion de projet en Section 6.3).

5.6. Un exemple d'exercice autour de la résolution de problèmes

L'exemple (pas mal plus ambitieux/complexe que les précédents) est inspiré d'un exercice de maths papier/crayon dont l'objectif 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.

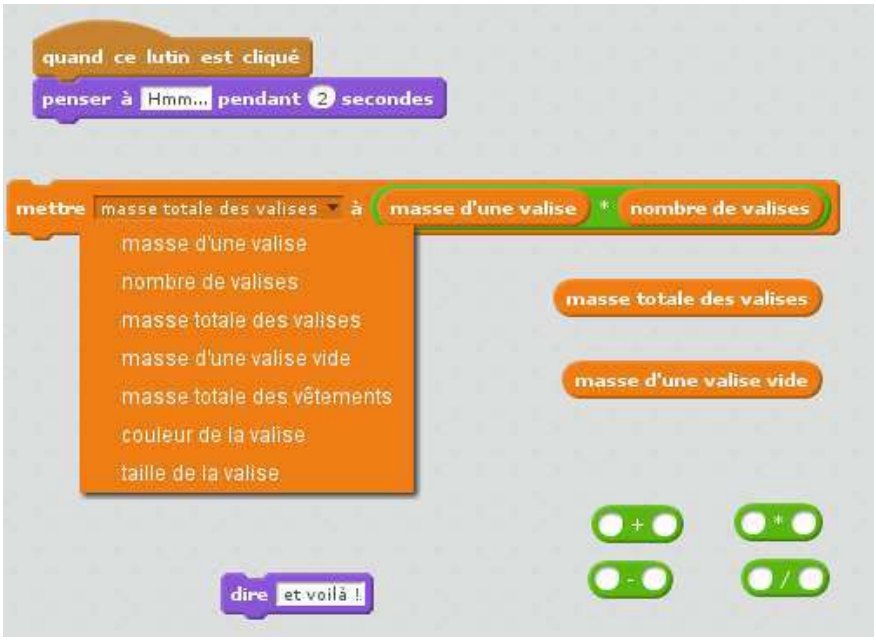
Scratch peut être utilisé pour réifier ce scénario de différentes façons, dont celle-ci : créer 3 personnages (Do, qui est chargée d'identifier les données, Val, qui est chargé d'identifier les valeurs à trouver, et Cal, qui est chargée de faire les calculs). Le 4^{ième} personnage, magicien / chef d'orchestre, 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.

<table>
<tr>
<td> 
</td>
<td> 

Le script du chef d'orchestre enchaîne 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).</td>
</tr>
</table>

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.

<table>
<tr>
<td> 
</td>
<td> 
</td>
<td>L'élève doit identifier les données (les variables) connues puis écrire le script (le programme) qui leur met leur valeur. Ici, l'élève est aidé par une liste de variables définies préalablement (il serait possible de laisser les élèves identifier et créer ces variables, mais cela suppose une bonne pratique déjà).</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
<td>L'élève doit identifier les valeurs (les variables) qu'il faut calculer. L'action « montrer » les affichera à l'écran.</td>
</tr>
</table>

		L'élève doit écrire les opérations permettant de faire les calculs.

Dans cette mise en œuvre, le point qu'il faut noter est que la structure du programme réifie la démarche de résolution de problèmes identifier les données / identifier les valeurs recherchées / effectuer les calculs. Ainsi décrite à un niveau abstrait, elle s'applique de façon générale (pour un autre exercice, il faudra simplement adapter les variables et les valeurs mises en jeu).

J'ai testé cet exercice avec des trois groupes de trois élèves de CM2. 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. De façon générale, cet exemple est surtout illustratif, sa mise en œuvre effective demanderait pas mal de travail pédagogique et de temps d'enseignement.

5.7. Analyse et 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, écrire une boucle, inventer un dialogue, etc. La tâche proposée peut varier d'un exercice très spécifique (par exemple, l'objectif étant de faire travailler la notion d'instruction conditionnelle, les faire compléter un programme qui nécessite un Si-Alors-Sinon) a un projet nécessitant une démarche complète (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 (cf. Section 6) relève, dans sa dimension algorithmique, du cas limite où les élèves définissent eux-mêmes le problème qu'ils considèrent.

Scratch peut 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). L'exécution au fur et à mesure de bouts de programmes permet de tester/évaluer les idées sur le mode essai-erreur (ce qui permet, du point de vue des compétences générales à aborder au primaire, de travailler le triptyque anticipation / contrôle / validation). Scratch peut également 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. Dans ce cas, il faut

faire attention à la correspondance entre le langage utilisé dans la phase papier/crayon et les blocs que propose Scratch. 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 hasard dans les blocs, ce qui serait contre-productif.

Lorsque l'on travaille la conception / l'écriture d'algorithmes directement sur machine, 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.

Que l'on passe par une étape papier/crayon 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 ».

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.

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).

Quelques remarques personnelles pour terminer :

1. J'aime bien l'idée d'utiliser l'algorithmique pour faire travailler la logique en tant que compétence transverse générale. En informatique, dès que l'on veut faire des instructions 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.).
2. 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.
3. 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 ? De fait, il y a différents avis. Mais la réponse est-elle vraiment importante ?

6. Comment aborder l'enseignement de la « pensée informatique » ?

Les exemples présenté en Section 5 illustrent 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[36] : une approche orientée « algorithmique » et une approche orientée « créativité »

6.1. Approche orientée algorithmique

Une première approche (et/ou conception) possible de l'enseignement de la pensée informatique est de l'aborder sous l'angle disciplinaire. Le point d'entrée est la discipline informatique telle qu'elle existe (dans l'industrie, dans la recherche), et est enseignée à l'Université. 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 (notions d'action, variable, boucle, conditionnelle, etc.).

Cette approche peut être plus ou moins liée à l'ancrage technologique que constituent les ordinateurs. Dans une mise en œuvre techno-centrée, l'objectif in fine est que les élèves sachent comprendre/faire des programmes, ce qui amène à les faire travailler pour (1) savoir faire des algorithmes et (2) savoir utiliser un langage de programmation. Dans une mise en œuvre plus centrée sur l'apprentissage de mécanismes d'analyse et de résolution de problèmes, des exercices d'algorithmique sans ordinateurs peuvent être utilisés. Ces deux visions ne sont pas nécessairement en opposition et peuvent cohabiter.

On peut noter que la plupart des arguments en faveur de l'enseignement de l'informatique et des propositions de programmes que l'on peut lire en France sont plutôt ou essentiellement sous-tendus par cette approche, cf. Section 2. On trouve également des choses similaires dans des textes produits par des associations pour l'enseignement de l'informatique aux USA[37].

6.2. Approche orientée créativité

Une seconde approche (et/ou conception) possible de l'enseignement de la pensée informatique et de la programmation est de prendre 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, la pensée informatique peut être abordée et enseignée comme une compétence relevant, en tant que telle et comme moyen, de ces « 21st Century challenges ».

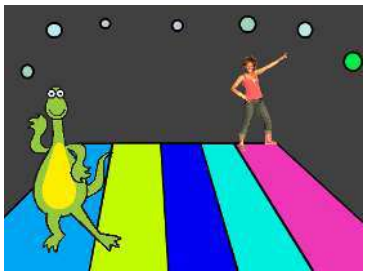
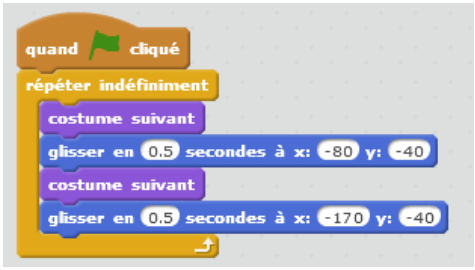
L'idée est de placer l'élève en situation de créativité et de proposer l'informatique comme un moyen pour exprimer cette créativité. L'objectif est que les élèves développent ainsi un lien personnel avec l'informatique, que ce soit pour eux une façon de s'exprimer (et, par suite, de penser, de comprendre, en faisant appel aux notions ou façons de faire de l'informatique). Cette approche met l'accent sur d'autres dimensions que l'algorithmique donc. Elle est généralement référencée comme « l'informatique créative » (traduction de « Creative Computing », qui est notamment le titre d'un livre-référence sur l'utilisation de Scratch)[38].

L'exemple ci-dessous (pris sur le site de Scratch) montre comment « faire danser deux personnages » amène rapidement à des problèmes qui nécessitent d'écrire des algorithmes (ici, une boucle), et mobilisent également les compétences de généralisation/réutilisation (les problèmes/solutions pour faire danser la fille et faire danser le dinosaure sont similaires).

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

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

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

		
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.		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).

Les exemples autour de la production d'écrit (Section 5.5) relèvent de cette approche. L'exemple introductif (imaginer un dialogue entre deux personnages) amène à écrire un algorithme simple et linéaire (pas de boucle, pas de conditionnelle : une simple juxtaposition d'actions). L'exemple « projet » consistant à amener les élèves à réaliser une histoire (« fiction interactive ») avec un schéma narratif complexe, différentes scènes (etc.) amène très vite au besoin (et c'est cela qui est important : le besoin) de savoir résoudre des problèmes (un peu) plus complexes, mobilisant des conditionnelles notamment (Si - Alors - Sinon).

De fait, les promoteurs de cette approche, sur la base d'études des projets construits par les élèves/enfants (en contexte institutionnel ou pas), montrent que ce type d'activité les amène à pratiquer/aborder les compétences d'algorithmique[39].

Deux points encore pour faire comprendre l'idée. Les promoteurs de cette approche ne parlent pas de faire faire des algorithmes ou des programmes aux élèves, mais de produire des « médias interactifs ». Et, sur une enquête où ils ont demandé à des enfants utilisant Scratch en contexte parascolaire 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[40], la notion de « créativité » et l'esprit général semblent assez différents).

6.3. Discussion

Avoir en tête ces 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) est, comme en maths ou d'autres domaines, de faire travailler les élèves sur un exercice identifié comme un vecteur d'apprentissage

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

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

de cette notion. Cependant, bien entendu, il est possible de développer cette stratégie en abordant des sujets sensés intéresser les élèves. Si l'on se place dans le cadre de l'informatique créative, le point d'entrée 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. Ceci étant, la façon d'aborder le domaine selon l'une ou l'autre approche/conception va amener à considérer différents critères, modalités et/ou priorités.

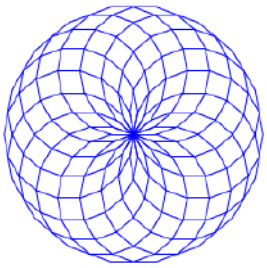
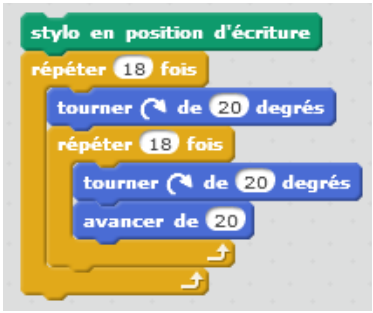
Pousser l'analyse plus loin nécessiterait de développer une réflexion épistémologique qui sort du cadre de ce document[41]. Je liste donc simplement quelques éléments de réflexion complémentaires.

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 (et 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 souvent 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)[42]. 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 cette conception de l'enseignement de la pensée informatique : 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 problèmes, etc.) associées à ce type d'activité, et pas uniquement pour programmer des algorithmes.

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[43].	Le programme en Logo.	Le programme en Scratch.

[41] 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 ?

[42] 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.

[43] Cet exemple illustre la possibilité d'utiliser Scratch pour la compétence « construction de figures » évoquée dans les programmes (il faut évidemment commencer par des figures plus simples, cf. les ressources pédagogiques Logo).

Quelques remarques personnelles pour terminer :

1. J'aime bien l'idée de travailler sur des projets où l'on passe 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) ; 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 d'écrits).
2. En passant, voici 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.). Prendre une grand feuille de papier Kraft et la fixer sur le mur (une feuille de papier cela se déplace, c'est mieux que d'utiliser un tableau fixe). Délimiter trois colonnes pour identifier les tâches « à faire » (tous les post-it sont là au début), « en cours » et « terminées». Faire des réunions collectives (par exemple au début et à la fin de chaque séance), debout devant ce support, pour distribuer les tâches (attribuer les tâches à des élèves ou des binômes et/ou laisser les élèves décider des tâches qu'ils-elles veulent réaliser, et ajouter le nom de l'élève sur le post-it par exemple) ; analyser régulièrement où l'on en est ; faire passer les post-it d'une colonne à une autre ; définir de nouvelles tâches ou décomposer une tâche trop complexe ; 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[44].
3. Toujours dans la veine « collaborative » : pendant les phases de programmation, toutes les x minutes, demander « qui a un problème qu'il ne sait pas résoudre ? » puis « qui sait faire cela ? » (*vs*. : c'est l'enseignant qui résout tous les problèmes).
4. Outre la dimension pluridisciplinaire, travailler sur ce type de projet peut également être utilisé pour des activités inter-cycles[45] : élèves de Cycle 3 qui produisent une histoire, un jeu ou un quiz pour des élèves de Cycle 2 ; élèves de Cycle 2 qui produisent des images qui servent de « scènes », les élèves de Cycle 3 travaillant sur les aspects algorithmiques ; dans le cadre de la continuité école/collège, travail collaboratif CM2/6ième ; etc.

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

Ainsi qu'indiqué précédemment, l'un des écueils de l'enseignement de l'informatique à l'école primaire est que les enseignants concernés n'ont, pour la plupart, jamais étudié l'informatique dans leur parcours de formation initiale[46]. 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 ». Nous avons vu en Section 4.2 que, pour ce qui est de l'algorithmique, le contenu à maîtriser est très limité. Dans cette section, j'essaie de donner quelques

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

[45] Cela peut également être le cas de certaines activités plus orientées « algorithmique » bien sûr.

[46] 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.

éléments de réponse pragmatiques pour Scratch (sans aborder la question, par ailleurs légitime, de la certification).

7.1. Pourquoi Scratch ?

Au niveau de l'école (et du collège), 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 :

1. 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).
2. 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.
3. 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 6).
4. 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[47]. Le fait qu'il y ait une communauté de pratique est un atout majeur.
5. Scratch 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 et absolument génial, tout juste inventé par deux passionnés »), c'est prendre le risque que dans quelques mois cela ne marche plus (les passions, cela passe ; derrière Scratch, il y a des institutions solides).
6. Il est possible d'utiliser Scratch en ligne via son site Web (pas d'installation à faire, possibilité de partager des ressources) et/ou de l'installer sur les ordinateurs de l'école (ce qui permet de travailler en cas d'absence ou de panne réseau).

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

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

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 de nouveaux 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 enseigner l'algorithmique, il faut une compréhension de Scratch en tant que langage informatique, i.e., en tant que moyen pour exprimer un algorithme (liste des actions à réaliser pour que le programme fasse ce que l'on souhaite). Pour cela, ce qu'il est nécessaire de connaître et de savoir utiliser a été présenté en Section 4.2.

Pour enseigner la pensée informatique dans une approche créative, il peut être utile de regarder un peu plus en détail les aspects multimédia (jouer des sons, créer des scènes ou des personnages à partir de photos, etc.) afin de pouvoir inspirer/guider les élèves. Il n'y a aucune nécessité de tout connaître, ni même

[47] 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.

de savoir en détail comment fonctionne quelque chose (la façon d'intégrer du son, de déformer un personnage, etc.) : il suffit d'essayer, ou de lire la documentation, ou de chercher des exemples faisant ce que l'on veut faire (…) quand la question se pose.

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 d'écrits, 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.

7.3. Comment s'y prendre ?

Première chose à faire, pour avoir une idée intuitive des choses : aller sur le site Scratch[48] (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[49].

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

1. Le document « Concepts de programmation et compétences développés avec Scratch »[50], qui synthétise (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.
2. Le guide référence (23 pages)[51], qui détaille l'environnement et le langage (les différents menus, la scène, les personnages et leurs costumes, etc.).

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

Il n'y a pas besoin d'être hyper-pointu, de connaître 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 maîtriser 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 connaître 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 des blocs et choisir le plus

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

[49] Attention cependant : le site Scratch propose (au moment où j'écris ce document) plus de 21 millions de projets. L'immense majorité sont sans intérêt, ni ludique ni pédagogique. Il ne faut pas s'arrêter au premier exemple venu donc.

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

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

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).

7.4. 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 :

1. 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.
2. 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 connaître, pour pouvoir faire la part des choses (savoir-faire cette part des choses est sans doute l'un des enjeux importants de la formation).
3. Un truc tout bête mais important : la possibilité de dupliquer des blocs par un clic droit (travail sur l'abstraction et la réutilisation).
4. 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.
5. 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.
6. Le fait de savoir définir soi-même des blocs simples (comme le « Avancer de 1 » de l'exemple de maths) 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.

8. 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 sont abordés : 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.

8.1. Exploiter les manuels autours de Scratch

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

1. Bien commencer avec Scratch, introduction à l'informatique (2013)[52], traduction française du livre « Starting from Scratch: an introduction to computing science ».
2. L'informatique créative (2011 puis 2013 ?)[53], traduction française du livre « Creative Computing Curriculum Guide ».

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

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

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 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 maîtrise du langage Scratch (e.g., maîtriser les blocs de la catégorie « Apparence » et l'éditeur d'images) et des objectifs de maîtrise de notions informatiques (e.g., la notion de répétition/boucle). La dimension algorithmique est moins prégnante que dans le premier livre.

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[54] ».

8.2. 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 :

1. Partir de ressources produites pour d'autres disciplines (cf. exemple des fractions en Section 5.3).
2. Partir d'analyses didactiques (cf. l'exemple de la numération en Section 5.4).
3. Reprendre ou s'inspirer des activités des manuels Scratch qui mobilisent la notion/compétence cible.
4. 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.
5. S'appuyer sur des activités/projets liés à l'histoire de la classe.
6. …

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).

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

9. 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[55].

1. Accès à des ordinateurs. Le fait d'avoir suffisamment d'ordinateurs pour travailler en classe entière peut sembler idéal … mais peut également poser des difficultés (cf. point suivant). 1 ou 2 ordinateurs de fond de classe (avec 2 ou 3 élèves par poste) permettent d'organiser des ateliers tournants et/ou des activités de délestage.
2. 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[56]).
3. 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 8 proposent des activités destinées à ce que les élèves découvrent/maîtrisent l'environnement.
4. 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 connaît pas toutes (mais qu'on peut explorer pour revenir, plus tard, avec une explication).
5. 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 qu'il fait effectivement progresser les élèves (ou certains élèves) sur certaines compétences plus que (ou différemment) d'autres activités pédagogiques.
6. 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).
7. 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

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

[56] 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'autonomisation.

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).

8. 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 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 nécessite 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/maîtrise 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 algorithmique » 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 réflexion.

- ✗ Quelles conceptualisations des notions sous-jacentes (y compris celle de « langage de programmation ») développent les élèves[57] ?
- ✗ …

10. 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 séquence ou une séance (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 séquences/séances.

[57] A titre d'exemple, indiqué par un enseignant : les élèves mettent des blocs Scratch dans la zone de programmation sans les lier entre eux (i.e., sans les structurer en un programme), puis cliquent sur les blocs les uns après les autres et obtiennent le résultat recherché : ce n'est pas un programme mais un ensemble d'instructions éparses non-structurées, cela marche sur des exemples simples, mais ce n'est plus le cas dès qu'il y a des structures complexes.