APPRENTISSAGES INFORMATIQUES ET USAGES DE LA VARIABLE INFORMATIQUE AVEC SCRATCH DANS LA CLASSE DE MATHEMATIQUES

Vanéa CHIPRIANOV [1]
Georges SALIBA [2]
Grégory TRAIN [3]

Résumé : Dans cet article, nous examinons la construction de connaissances informatiques dans l'environnement spécifique que constitue Scratch. Nous prenons appui en particulier sur une expérimentation conduite en classe au cycle 4 que nous analysons au filtre des différentes dimensions que nous présentons en amont. Il s'agit *in fine* d'examiner les potentialités et éventuelles contraintes de l'environnement Scratch dans la construction plus globale d'une expertise informatique en classe de mathématique.

Introduction

Le paysage scolaire de l'algorithmique et de la programmation s'est fortement densifié dans les nouveaux programmes de 2016 en diffusant désormais de l'école jusqu'au lycée. Cette entrée, étirée sur un curriculum particulièrement long, exigeant à la fois une mise en cohérence et une progressivité des contenus enseignés, s'est accompagnée de l'arrivée de nouveaux outils logiciels, dans lesquels Scratch occupe une place de choix notamment aux cycles 3 et 4. Dans le même temps, ce sont les enseignants de mathématiques qui se sont vus confier au moins pour partie la prise en charge de ces enseignements. C'est donc un paysage redessiné dans une certaine mesure et des questions d'enseignement et d'apprentissage de l'algorithmique et de la programmation partiellement renouvelées qui se posent aujourd'hui.

L'accompagnement institutionnel de ces changements a pris (et continue à prendre) différentes formes (stage, formation à distance, etc.). Différents documents ressources ont été en particulier adressés récemment aux enseignants de mathématiques de collège et de lycée. Sans entrer dans le détail de ces derniers, un soin particulier semble avoir été pris dans l'examen des différences entre variable informatique et variable mathématique, en prise d'appui sur une expertise bien assise des enseignants de mathématiques concernant les difficultés de conceptualisation afférentes à la variable mathématique.

1 Université de Bretagne Sud, IRISA, Vannes, France
2 C2i TICE, Collège Aliénor d'Aquitaine, Bordeaux.
3 C2i TICE, Laboratoire Lab-E3D (EA 7441), université de Bordeaux.

Si cette documentation semble bien outiller la profession dans la prise en compte de la distance entre variables mathématiques et informatiques, son corollaire, nous semble-t-il, est celui d'une définition « en creux » de la variable informatique et à tout le moins, d'un pôle variable informatique et des difficultés spécifiques liées à sa conceptualisation moins investies.

Nous analysons dans cet article la construction de connaissances dans les domaines mathématique et informatique en proposant conjointement d'identifier les difficultés de conceptualisation, d'usages et de mise en œuvre des variables dans ces deux domaines. Ce travail est conduit dans un premier temps à travers l'analyse d'exemples illustratifs. Dans un second temps, l'étude de situations proposées en classe de mathématiques permettent d'enrichir et de compléter de telles analyses. Nous outillons nos analyses en prise d'appui sur les travaux de Hoc (1984) et Rogalski (1998) lesquels permettent de distinguer a priori différentes trajectoires possibles dans le processus de programmation conduit par un programmeur. Un tel point de vue permet en retour d'investir la diversité des techniques de résolution possibles de problèmes. Cette diversité donne alors accès à des usages spécifiques des variables en jeu dans l'activité de programmation. Nous montrons par ailleurs que les caractéristiques de l'environnement Scratch, en particulier le paradigme événementiel dans lequel il s'ancre, enrichissent cette diversité.

Activité de programmation & traitement d'un problème

Si l'on veut bien s'accorder sur le but de l'activité de programmation, à savoir la production d'un texte exécutable sur un dispositif informatique particulier, force est de constater que ce seul but lui confère des spécificités, en particulier au regard des voies possiblement empruntées pour l'atteindre. Hoc (1984) puis Rogalski (1998), en regardant la trajectoire d'un programmeur comme partant d'un problème sur les « objets du monde » pour arriver à la production d'un texte de programme exécutable sur un dispositif informatique, proposent de distinguer deux voies possibles, lesquelles engagent différemment les compétences du programmeur.

— Une première voie consiste à résoudre dans un premier temps le problème à la main (phase de traitement du problème) puis à opérer l'analyse de ce traitement pour le traduire en une solution informatique exécutable (phase de représentation du traitement). Dans cette perspective (Hoc, 1984), le travail de représentation du traitement nécessite d'exprimer les différentes étapes de la résolution à la main dans un langage informatique donné.

— Une seconde voie consiste, quant à elle, à débuter par la représentation du problème dans le monde des objets informatiques (phase de représentation des données). Ce travail engage alors prioritairement l'identification des variables informatiques susceptibles de s'organiser en un système permettant de représenter le problème. Il s'agit alors de penser la description et l'opérationnalisation des changements d'états de ce système (au moyen des opérations élémentaires du système informatique supportant l'implémentation de la solution), ingrédients d'un travail qui relève de la phase de traitement du problème.

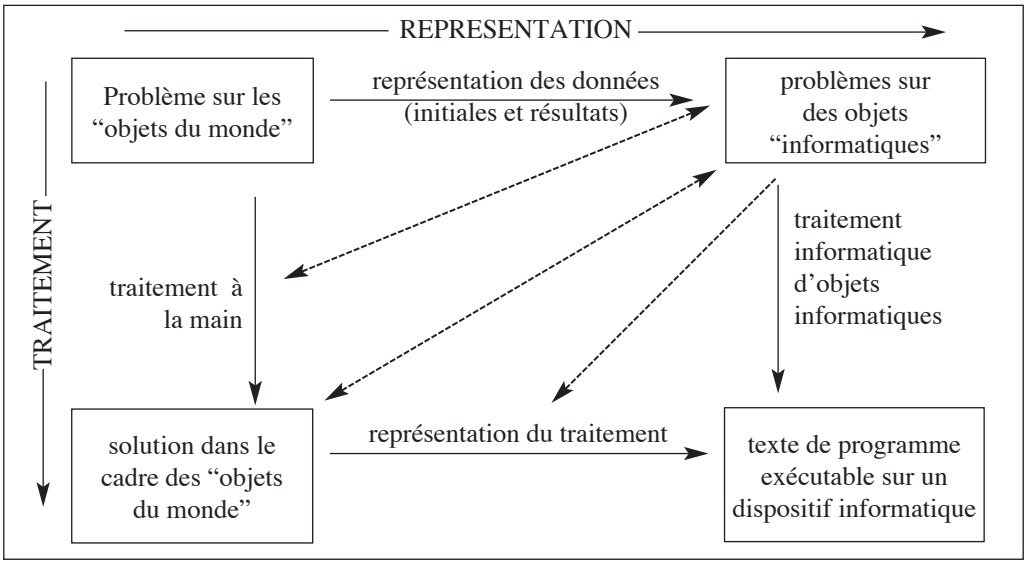


Figure 1

Le schéma (Figure 1) proposé par Rogalski (1998, p.66) rend compte de ces deux voies possibles. Si un tel schéma permet de dessiner a priori deux voies clairement distinctes, il donne à voir également une réalité plus complexe dans les trajectoires empruntées, et des interrelations possibles au cours de chacun des processus. Ainsi, de façon dynamique, l'avancé du traitement à la main du problème pourra influencer le choix des variables informatiques. De la même façon, le choix d'une représentation des données permettra de reconsidérer le traitement à la main au cours du processus.

C'est l'expertise du programmeur [4] qui pilote l'intensité de ces interactions. Cette expertise s'exprime au niveau de la connaissance du fonctionnement interne du système informatique, en particulier de la connaissance des opérations compatibles avec le dispositif informatique.

Dans le même temps, ce sont des connaissances spécifiques au domaine dans lequel le problème se situe - pour notre cas, des connaissances mathématiques - (et/ou leurs besoins) qui vont s'exprimer inégalement dans l'emprunt de l'une ou l'autre des voies possibles. Ce sont ces mêmes connaissances mathématiques qui vont également participer de l'intensité des interactions.

Dans cette section, à partir de l'exemple de « l'implémentation du critère de divisibilité par 9 », nous identifions les connaissances mathématiques et/ou informatiques nécessaires à l'implémentation de la décomposition d'un nombre en chiffre pour en faire la somme. Une telle implémentation suppose de déterminer et/ou choisir une représentation pour des grands nombres non nécessairement représentés par

4 Les études conduites par Rogalski s'ancraient en partie dans une dimension comparatiste, entre programmeurs expérimentés et programmeurs novices.

des types préexistants dans les langages de programmation. Il s'agit d'implémenter un traitement du critère de divisibilité par 9 pour la représentation choisie. Dans l'une et l'autre des voies susceptibles d'être empruntées dans l'activité de programmation suggérées par Rogalski (Figure 1), les connaissances potentiellement en jeu diffèrent et convoquent des usages de la variable mathématique et informatique distincts.

Les deux programmes présentés ci-contre (Figure 2) s'emploient l'un et l'autre à déterminer si un nombre entier donné est divisible par 9. Ils se distinguent par ailleurs en illustrant l'une et l'autre les deux voies possiblement empruntables et proposées par Rogalski.

Plus dans le détail, la saisie d'un nombre par l'utilisateur et sa mémorisation dans la variable *nb* étant réalisées, les deux scripts illustrent deux expressions différentes du traitement du nombre mémorisé dans la variable *nb* pour en faire la somme de ses chiffres. Le script déclenché par l'envoi du message *test* permet quant à lui, de réitérer le processus dès que la somme des chiffres est supérieure strictement à 9. L'ensemble du programme est donné en annexe n°1.

Le premier script (Figure 2) entretient une proximité forte avec la procédure « à la main » du test de divisibilité par 9, dans la mesure où l'intention première est celle de faire la somme des chiffres du nombre en jeu, sans se préoccuper en amont de la possible traduction de cette procédure au sein du dispositif informatique. La traduction de cette procédure « à la main » compatible avec le dispositif engage alors la recherche d'opérations compatibles avec le dispositif. Ici, les opérations effectuées pour extraire les chiffres composant le nombre saisi sont des opérations sur des chaînes de caractères, lesquelles sont utilisés pour l'affectation de la

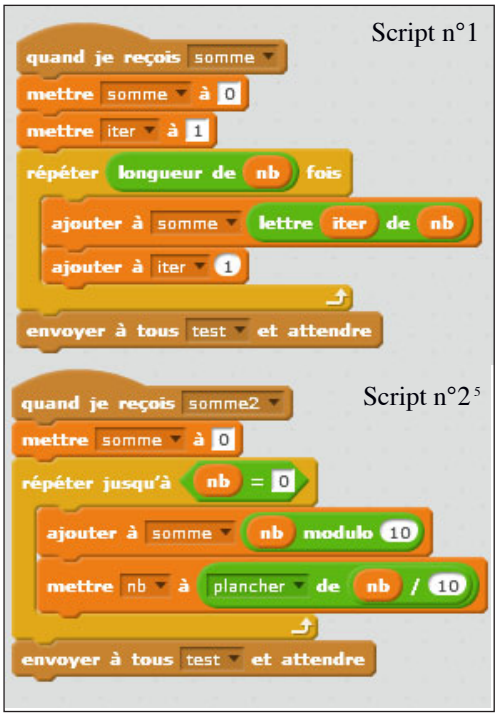


Figure 2

variable *somme* et lors de l'évaluation de l'expression produite.

Dans le second script (Figure 2), la voie empruntée est sensiblement différente, avec une intention première de représentation du

5 La fonction « plancher » renvoie la partie entière d'un nombre fourni en argument.

problème dans le monde des objets informatiques. Deux opérations arithmétiques sont ainsi retenues pour permettre de sommer les chiffres du nombre saisi et ce sont des entiers qui sont utilisés dans les expressions. Il s'agit de chercher à décomposer, chiffre par chiffre, un nombre contenu dans une variable de type entier, ce qui nécessite d'exprimer arithmétiquement une telle décomposition.

Dans ce premier exemple, particulièrement simple, apparaît une expertise du programmeur inégalement investie dans l'emprunt de l'une et l'autre des voies. La production du premier script n'est rendue possible qu'à la condition de disposer d'un répertoire d'opérations compatibles avec le dispositif informatique. L'analyse de ce même script souligne qu'au-delà des voies strictement empruntées, ce sont avant tout dans les interactions entre le traitement à la main et le choix d'objets informatiques représentant les données que se joue la production du script, interactions pilotées par la connaissance d'un type de variable informatique et des méthodes qui lui sont associées dans l'environnement [6].

Ainsi, et quand bien même, comme nous l'avons par ailleurs souligné, l'une et l'autre des voies sont *in situ* susceptibles de variations, porter un regard *a priori* sur l'une et l'autre de ces directions possiblement empruntées permet d'exhiber les possibles difficultés à gérer la conceptualisation de la variable informatique, dans la mesure où une telle posture permet d'outiller le regard sur la diversité et/ou proximité des multiples traitements des variables mathématiques et informatiques potentiellement en jeu dans une telle activité de programmation. Dans le même temps, un tel point de vue met à jour des connaissances mathématiques inégalement investies. Quand, dans le second script, des connaissances sur la division euclidienne et sur la troncature sont pleinement sollicitées, le besoin en connaissances mathématiques dans le premier script apparaît faible, alors même que ce dernier semble plus proche d'une procédure « à la main » du test de divisibilité par 9 pour lequel on ne cherche pas à expliciter les opérations permettant d'extraire les chiffres du nombre testé. Plus encore, c'est le regard même à poser sur le critère de divisibilité par 9 qui se distingue dans l'un et l'autre des deux scripts. Notons enfin, concernant l'environnement de programmation, que si des modifications de type de variable opèrent dans le premier script (passage d'un caractère à un entier dans l'opération arithmétique « ajouter à » [7]), ils peuvent apparaître transparents pour le programmeur, car gérés avec la même transparence par l'environnement, ce qui relève d'une spécificité de Scratch contrairement à d'autres environnements de programmation. Ce dernier point est alors d'importance dans la prise en compte des possibles interactions entre traitement à la main et variétés des représentations des données au sein du dispositif.

Ce premier exemple détaillé illustre, nous semble-t-il, toute la fonctionnalité du schéma proposé par Rogalski, permettant de « situer » la trajectoire du programmeur au regard des connaissances mathématiques et informatiques ici les variables déployées dans la production d'une solution. A ce titre, le script n°1 peut être considéré comme la représentation des données pour traitement sur la représentation informatique du problème et le script 2 comme l'expression mathématique du traitement à la main qui est ensuite représentée pour le dispositif informatique. Dans ces deux scripts, la variable

6 En ce sens, il est délicat de statuer, à partir de ce seul script, s'il relève strictement de la représentation du traitement à la main ou d'un choix de représentation des objets du problème.
7 Ici l'environnement Scratch associe au caractère 9 la valeur 9 ce qui n'est en général pas le cas puisqu'en général c'est la valeur qui code ce caractère dans la norme d'encodage sous-jacente qui est retournée.

« *nb* » est d'un type informatique différent. Dans le premier, c'est une chaîne de caractères et elle reste constante durant toute l'exécution du script tandis que dans le second, cette variable est entière et elle est modifiée à chaque passage dans la boucle. Ce sont ces usages distincts qui permettent d'affiner le regard à poser sur les variables et les difficultés relatives à leur mise en œuvre.

Aussi, l'expertise du programmeur et en particulier l'évolution de sa propre représentation du fonctionnement interne du système informatique, participe des différentes voies d'investigations d'un problème donné. Parce que la voie privilégiée par les programmeurs débutants est celle d'exprimer et de transcrire de manière assez directe le traitement à la main du problème dans l'environnement de programmation, la question des diverses possibilités de représentation par des objets informatiques des objets du « monde réel » au sein de l'environnement Scratch se dessine comme centrale. Nous investissons cette voie dans la section suivante en précisant brièvement dans un premier temps le point de vue adopté sur la construction de l'expertise d'un programmeur. Ceci nous permet dans la suite d'examiner le logiciel Scratch et ses spécificités dans la construction de cette expertise.

Expertise du programmeur

Pour Samurçay & Rouchier (1985), il s'agit, dans les situations de programmation, non seulement de produire un résultat en réponse à un problème donné mais également d'expliciter la procédure de résolution de ce problème pour un dispositif informatique. Il s'agit en somme de passer du « faire » au « faire faire » en établissant notamment, du côté du programmeur, un plan d'actions dont l'exécution est différée et déléguée à une machine. L'expertise du programmeur, dans une telle tâche, se joue pour Hoc (1977), au niveau de la construction, propre à chaque programmeur, de son système de représentations et de traitement informatique (SRT), « *résultat de l'intériorisation du langage de programmation, de ses modes d'utilisation et des procédures couramment employées dans le domaine d'activité en question* » (Hoc, 1977, p.16). A l'instar de nombreux auteurs (Viallet & Venturini, 2010 ; Nivat, 2009 ; Baron et al., 2011, Dowek, 2005), nous partageons l'idée que le domaine « algorithmique et programmation » est au cœur de la discipline informatique et forme un tout indissociable dans un enseignement de l'informatique. Adopter un tel point de vue [8] permet de regarder l'activité de programmation comme permettant d'enrichir dialectiquement le SRT d'un programmeur : l'implémentation en machine d'un problème donné participe à la construction du SRT du programmeur, en l'enrichissant potentiellement d'opérations et d'instructions élémentaires permettant la planification d'étapes dans la résolution d'un problème. Dialectiquement, cet enrichissement progressif permet de penser en retour les possibles implémentations machine et de capitaliser ainsi des connaissances attenantes aux spécificités de ces implémentations.

En Python, l'échange du contenu de deux variables s'exprime directement avec l'instruction « a, b = b, a ». Cette solution s'appuie sur la connaissance d'affectations « simultanées » de plusieurs variables dans ce langage de programmation.

Avec Scratch (Figure 3), plusieurs instructions (et éventuellement une variable temporaire) sont nécessaires pour implémenter cet échange.

8 Un tel point de vue tranche avec la promotion de l'informatique débranchée, qui nous semble tenir faiblement compte de cette dialectique dans la construction de l'expertise d'un programmeur.

Figure 3

Réduire la pratique de la programmation à celle de l'enseignement d'un langage, c'est sur-exprimer une seule visée utilitaire au détriment d'une visée plus large. Cette visée est celle de l'enseignement des langages de programmation comme moyen d'enrichir le système de représentations et de traitements du programmeur, d'augmenter sa capacité à structurer en systèmes les opérations informatiques élémentaires, au service du but de son activité.

Logiciel Scratch & construction de l'expertise du programmeur

Scratch est un environnement de programmation par bloc qui permet d'exécuter des instructions déclenchées par des événements (les blocs chapeaux, le double clic, les messages). Chaque jeu d'instructions attaché à un évènement constitue un script. Les scripts sont quand à eux attachés à des lutins ou à la scène. Ces derniers sont des objets qui ont leurs attributs propres (l'ensemble des variables qui les caractérisent), comme la direction, la position etc. Ces objets ont par ailleurs des méthodes, au sens de la programmation orientée objet (un ensemble d'instructions/fonctions), qui permettent de modifier l'état du lutin ou de la scène et ainsi agir sur les attributs qui caractérisent chaque objet. L'environnement Scratch place ainsi le programmeur dans le paradigme de la programmation événementielle.

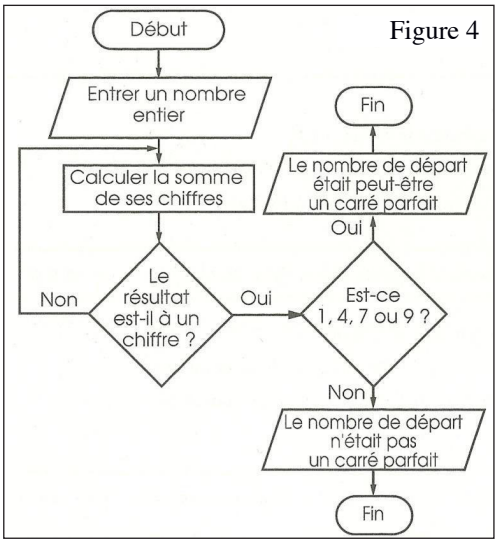
Parce que d'une part l'environnement Scratch est le premier environnement (à tout le moins à suivre les recommandations officielles) de programmation rencontré par les élèves apprentis programmeurs, parce que d'autre part, ces mêmes apprentis programmeurs sont susceptibles dans leurs premières activités de programmation de privilégier la voie de programmation du traitement à la main vers une solution exécutable en machine, il apparaît nécessaire de s'interroger sur les potentialités de Scratch relatives aux différentes implémentations d'un même problème, sur les connaissances associées à l'environnement (notamment en terme de contraintes liées à la syntaxe et à la grammaire du langage), ceci en lien avec la construction de l'expertise du programmeur. Nous empruntons cette voie à partir d'un exemple d'activité de programmation que nous filons tout au long de cette section.

Dans ce premier exemple, nous nous intéressons au caractère « être un carré parfait » des nombres. Il s'agit d'identifier si pour un grand nombre donné il est nécessaire de se lancer dans la rechercher d'un possible antécédent entier naturel sans entrer dans une démarche exhaustive plus couteuse. Dans cette première section, nous illustrerons les potentialités de l'environnement Scratch et les connaissances afférentes.

Les ingrédients d'une solution (identification des transformations, des tests, des données d'entrée et de sortie, planification des actions …) sont présentés dans l'algorigramme (Figure 4) de la page suivante.

Un programmeur expert identifiera que l'expression de la solution convoquera l'usage d'une boucle pour le calcul de la somme des chiffres qui composent le nombre courant et une autre pour éventuellement réitérer ce calcul avec la somme obtenue, jusqu'à obtenir un nombre à un seul chiffre. Au sein de l'environnement Scratch, différentes structurations de la solution sont possibles, notamment des solutions implémentées proches du traitement à la main du problème (Figure 5). Dans la solution implémentée dans l'environnement Scratch, le

Figure 4



concept de *message* est utilisé. Spécificités du logiciel, les messages servent à déclencher des scripts spécifiques.

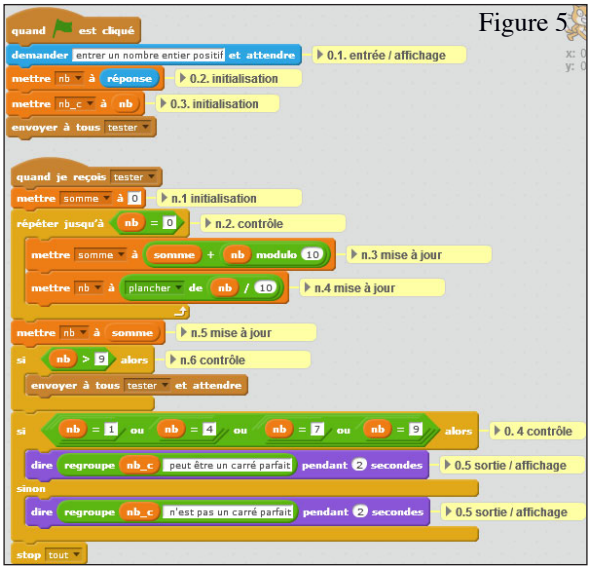
Ils peuvent également servir à déclencher des sous-scripts qui sont des instructions communes à plusieurs scripts. Ce concept de *message* permet en outre une implémentation du problème proche du traitement à la main du problème, dans la mesure où il permet une forme de communication entre scripts et donc une forme de prise en compte d'une temporalité des actions à conduire dans la résolution du problème. Cette trajectoire de programmation est par ailleurs plus volontiers empruntée par les programmeurs novices. Reste que l'implémentation d'une telle solution recouvre en réalité une forme de complexité attachée à des connaissances fines de l'environnement. Implémenter une telle solution nécessite ainsi de comprendre que l'instruction « *envoyer à tous tester et attendre* » va avoir pour trois effets qui sont chronologiquement ordonnés :

1. L'interruption du déroulement du script dans lequel cette instruction est exécutée
2. Le déclenchement de l'exécution du script attaché à la réception du message « tester »
3. La reprise de l'exécution du script qui a envoyé le message là ou il avait été interrompu.

Ainsi dans notre situation cette instruction permet lors de son exécution d'interrompre le déroulement du script courant et ainsi de le relancer depuis le début en attendant la fin de l'exécution de ce nouveau jeu d'instructions. L'environnement Scratch crée l'événement associé au déclenchement de ce script. La partie « *et attendre* » dans cette instruction permet de temporiser jusqu'à la fin de l'exécution des instructions associées au nouveau script sollicité. L'effet produit est donc le même que celui d'une boucle, la sortie de la boucle étant assurée par la variable globale *nb*.

L'instruction « *envoyer à tous et attendre* » révèle également une forme de complexité

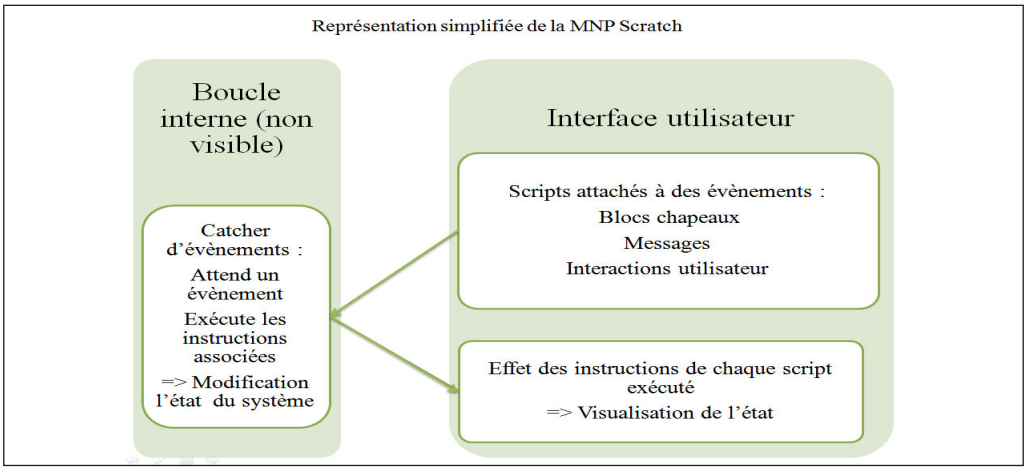
Figure 5



d'usage. Avec l'instruction « *envoyer à tous tester et attendre* », l'état final de la variable test sera 1 si l'utilisateur saisit par exemple 279. En revanche, en la substituant à l'instruction proche « *envoyer à tous tester* » l'état final de cette même variable sera 2 pour le même nombre (279) saisi. Ce sont en réalité des connaissances associées à la gestion de deux mêmes instructions après « *attendre* » qui sont en jeu ici : dans le moteur interne de Scratch, le principe d'exclusion mutuelle est appliqué pour ne pas autoriser la modification de la même variable au même moment. Plus largement, le recours au concept de message dans une telle implémentation, permettant des appels de scripts, nécessite de s'être forgé une représentation de l'effet d'un message durant l'exécution de scripts (en particulier l'effet d'interrompre l'exécution d'un script et de se comporter comme des événements). C'est une représentation dynamique de la machine qui est ici à l'œuvre (Figure 6), dans laquelle un script est attaché à plusieurs déclenchements d'un même événement et dans laquelle une boucle infinie, fonctionnant en arrière-plan dans la machine, permet de vérifier ces déclenchements.

Ainsi, ce que montre cet exemple est d'abord une propension à l'implémentation de problème proche d'un traitement à la main. En cela, cette perspective s'inscrit dans la trajectoire de programmeurs novices et est parfaitement conforme aux intentions des concepteurs du logiciel. Dans le même temps, cet exemple souligne que sans entrer dans la connaissance fine du concept de message, la possibilité de donner corps à la structuration de boucle n'est pas assurée. C'est un phénomène de transparence des apprentissages qui est possiblement à l'œuvre et que nous résumons elliptiquement par la formule « programmer sans conceptualiser ». A tout le moins, ces faits sont pour nous des éléments permettant à la fois d'orienter les choix de projets informatiques à proposer aux élèves mais aussi d'analyser les difficultés éprouvées par les élèves dans la conduite de tels projets. C'est ce que nous illustrons dans la suite de l'article. Avant de s'engager dans cette voie, nous examinons en amont les variables, leurs usages et leurs rôles potentiellement convoquées au sein du logiciel Scratch dans la perspective qui est la nôtre des difficultés d'acquisition de concept et plus largement de la construction de l'expertise du programmeur.

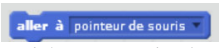
Figure 6



Logiciel Scratch et conceptualisation de la variable informatique

Considérer la variable informatique comme une boîte avec une étiquette n'est pas suffisant. Dans ses travaux Samurçay identifie deux types de variables « les variables qui sont des données explicites du problème (*variable exogène*) sont mieux traitées par les élèves que celles qui sont rendues nécessaires par la solution informatique et qui n'ont pas d'équivalent sémantique par rapport à la solution à la main (va*riable endogène*) ». (Samurçay R. , 1988, pp. 219-220). Dans notre étude nous en introduisons un troisième au regard des spécificités de l'environnement Scratch, les *variables prédéfinies* de l'environnement Scratch. Ces dernières sont à distinguer des deux autres types puisque d'une part elles ne sont pas déclarées par le programmeur et que d'autre part on n'opère pas toujours de la même manière sur ces dernières.

Parmi ces variables, on trouve d'une part les attributs des lutins (position, direction) qui sont des variables visibles par tous les lutins mais modifiables uniquement par le lutin dont elles dépendent. Et d'autre part, les valeurs retournées par les divers capteurs qui sont visibles par tous les lutins (par exemple *réponse*, *bord touché*).

Il est toujours possible d'interroger l'état de ces variables mais leur modification n'est possible que par une interaction (lutin, souris, scène) ou par l'intermédiaire d'instructions spécifiques du lutin lui-même, par exemple 

(cette instruction est locale) permet le déplacement d'un lutin et met à jour les variables qui définissent la position du lutin de manière transparente pour l'utilisateur.

Ce type supplémentaire n'est pas nécessairement d'intersection vide avec les deux autres types qui eux sont disjoints. Toutefois, cette catégorisation permet un nouveau niveau de lecture des difficultés que peuvent rencontrer les jeunes programmeurs.

Dans l'exemple étudié, les trois types de variables sont présents.

- *nb* est une donnée explicite (variable exogène) du problème lors de son initialisation et sera transformée puis éventuellement réinitialisée lors du traitement du problème (si la somme obtenue est supérieure strictement à 9, cette dernière remplace le nombre saisi).
- *nb_c* qui est une copie du contenu de *nb* pour permettre de ne pas perdre le nombre saisi lors de la sortie (variable endogène).
- *réponse* qui est une variable prédéfinie de l'environnement Scratch capture la saisie utilisateur (variable prédéfinie).
- *somme* est une donnée explicite du problème qui sera initialisée et mise à jour (variable exogène).

Ce premier niveau de lecture, de la mise en œuvre des variables, outillé par les opérations effectuées (mise à jour, test, contrôle, etc. (Figure 5) et les types de variables en jeu dans la solution informatique ne rend pas compte de l'intégralité de la complexité. Un second niveau de lecture est donné par le rôle que peut avoir une variable. Chaque rôle étant possiblement associé à des usages qui renvoient à des opérations. Concernant la variable « *nb* », un phénomène particulier est en jeu, elle porte le rôle de l'inversion du contrôle. En effet si la programmation séquentielle est structurée avec un ordre d'exécution déterminé (un seul flux de contrôle) la programmation événementielle requiert, quant à elle, une série d'appels à des petits scripts (handlers), qui fragmentent la logique du programme. Le flux de contrôle entre ces différents scripts n'est pas évident :

un programme enregistre avec l'environnement d'exécution son intérêt d'être repris (resumed) quand certains événements se produisent; c'est l'environnement d'exécution qui appelle un script quand un événement se produit ; ce n'est pas le programme qui appelle en clair les scripts.

Ainsi, le flux de contrôle entre les scripts n'est pas exprimé explicitement, mais seulement implicitement, en utilisant des variables globales, partagées entre les différents scripts. Dans notre cas ; c'est la variable « *nb* », qui est une variable globale, partagée entre les exécutions (multiples) du script attaché à la réception du message « *tester* » qui joue ce rôle. En effet, dans un premier temps, elle aura la valeur introduite par l'utilisateur – « *réponse* », ensuite elle prendra la valeur de la somme de ses chiffres. L'opération de mise à jour « *mettre nb à somme* » exécutée lors du n-ième appel du script influencera la valeur de « *nb* » lors de l'appel suivant de ce script. En ce sens, elle est globale et partagée entre les différentes exécutions/instances du script. De plus ce sont ces valeurs qui seront utilisées pour décider de l'envoie du message « tester » (si $nb > 9$ ») ou arrêter l'exécution. Dans le cas présent, ce sont donc bien les valeurs de la variable « *nb* » qui permettent de contrôler le flux, c'est donc un nouveau rôle de la variable informatique qui est en jeu, dont l'utilisation est plutôt complexe et possiblement source d'obstacle. Dans le même temps cette complexité est aplanie par l'environnement Scratch.

Si l'aspect dynamique des opérations qui se réfère à l'état du système et aux modifications que chaque instruction y apporte permet une première lecture des rôles et usages des variables dans cette implémentation, l'aspect invariant des relations entre *somme* et *nb* lors de l'exécution des instructions n.1 à n.6, en revanche, est source de complexité. Plus dans le détail, en notant $N_0 = (n_p n_{p-1} \ldots n_1 n_0)_{10}$ le contenu de *nb* exprimé en base 10 avant la boucle répétée jusqu'à… il s'agit de construire par récurrence finie, la fonction f définie, pour tout $N_0 \quad \mathbf{N}$, par : $f : N_0 \rightarrow \sum_{k=0}^{p} n_k$, avec $N_0 = (n_p n_{p-1} \ldots n_1 n_0)_{10}$. L'indentification d'une relation d'invariance permet de construire cette fonction pour tout $1 \leq k \leq p$:

$somme_k = \sum_{i=0}^{k} n_i = somme_{k-1} + N_k$ ([10])

et $N_k = (n_p n_{p-1} \ldots n_k)_{10} = [N_{k-1}/10]$ Les mises à jour n.3 et n.4 consistent en l'expression de ces relations au rang $k + 1$.

Exhiber une telle complexité ne relève évidemment pas, pour nous, d'une fin en soi. En revanche, elle permet d'outiller l'analyse de la complexité de la mise en œuvre d'une solution informatique au regard de l'usage des variables. L'exemple filé dans cette première partie montre ainsi qu'à la seule notion d'état des variables, l'aspect dynamique de la machine, qui relève de spécificités de l'environnement Scratch est nécessaire pour construire une expertise globale. Cela rejoint le point de vue de Samurçay (1998) relativement au concept de boucle et à sa conceptualisation. Dans la structuration d'une boucle, les variables interviennent de trois manières distinctes et imbriquées : l'initialisation, le test d'arrêt et la mise à jour. Ainsi c'est donc dans l'usage d'une boucle que le concept de variable informatique est le plus difficile à conceptualiser, la structuration d'une boucle convoquant les trois usages mentionnés précédemment et leur mise en relation. Dans l'exemple détaillé précédemment l'environnement logiciel Scratch permet de rendre une boucle externe avec le concept de *message*[9]. Pour autant la mise en œuvre d'une telle solution informatique relève de connaissances fines de l'environnement.

9 Alors que dans un environnement de programmation séquentielle, l'imbrication de deux boucles ou l'appel à la récursivité auraient été nécessaires

Nous pouvons évidement voir cette potentialité comme possiblement réductrice des difficultés en permettant de scinder et de rendre apparents les schémas de la planification à la main de la résolution d'un problème, ceci au prix de connaissances opérationnelles expertes relatives à l'environnement. Elle est pour nous, à tout le moins, des éléments d'analyse des projets informatiques proposés aux élèves, ce que nous présentons dans la suite de l'article.

Finalement, ces premiers exemples illustrent plusieurs niveaux de lecture relatifs aux difficultés d'usages de la variable informatique et mathématique : les types de variables en jeu dans le problème (endogènes, exogènes et prédéfinies), les opérations sur les variables (mises à jour, entrée, contrôle, etc.) leur réciprocité mathématique/informatique et leur rôle dans le programme (constante, itérateur, contrôle du flux, etc.) en plus de l'incidence de connaissances relatives à l'environnement Scratch. Ce sont ces aspects que nous mettons en fonctionnement pour l'analyse de projets en classe.

Regard sur un projet informatique en classe : les visages

Nous proposons dans cette dernière partie d'analyser un projet informatique proposé à des élèves de cycle 4 (cinquième & quatrième). Avant l'abord de ce projet, tous les élèves avaient fréquenté une première fois l'usage de variables dans l'environnement Scratch. La consigne donnée aux élèves est présentée dans l'encadré de la page ci-contre.

Eléments d'analyse du projet : l'activité de programmation

En prise d'appui sur la modélisation de l'activité de programmation de Rogalski (Figure 1), dans le projet « les visages », le choix a été fait d'imposer la représentation du problème aux élèves afin de recentrer l'activité sur l'expression du traitement des objets informatiques pour résoudre le problème.

Dans le détail, dans une première partie, la compréhension du choix de représentation des données est visée. Chaque visage est ainsi constitué d'un contour, d'une bouche, d'un nez et d'une paire d'yeux tous interchangeables. Les élèves disposent d'une liste exhaustive de nez, de bouches, d'yeux et de contours. Ce nombre important de données (et donc la possibilité de dessiner une multiplicité de visages) doit permettre de pointer la nécessité d'automatiser le processus de construction (en particulier de rejeter toute procédure consistant à considérer un jeu de données particulier). Le choix des valeurs ponctuant la liste (1000 ou 2000), hors de la fenêtre d'affichage de Scratch permet aux élèves d'identifier une mauvaise implémentation dans la mesure où le lutin sortirait de la fenêtre d'affichage. Dans la seconde partie, un nombre d'instructions restreint est proposé pour dessiner un visage. Le nom et le contenu de la variable prédéfinie « *réponse* » est affichée dans la scène de l'interface (case cochée). La troisième partie, à l'aide du concept de message, met en jeu des transformations géométriques (translation et homothétie de rapport 2 et de centre l'origine du repère pour les 4èmes et symétries axiales et translations pour les 5èmes) et leur expression algébrique.

Eléments d'analyse du projet : le fonctionnement spécifique du logiciel

Dans le cadre de la programmation séquentielle, un programmeur expert identifiera que l'expression de la solution convoquera l'usage d'une boucle non bornée et de deux instructions conditionnelles imbriquées ou non. Les instructions conditionnelles pour disjoindre les cas et la boucle non bornée pour permettre de poursuivre la saisie jusqu'au marqueur de fin,

I — Principe

On se propose de construire des visages à partir d'une base de données permettant de coder 5 bouches, 5 nez, 5 paires d'yeux et 5 contours.
Par exemple, pour coder la bouche numéro 1, dont le dessin est réalisé à partir de 8 points, dont les coordonnées sont :
(43 ; 25), (47 ; 32), (45 ; 29), (62 ; 29), (60 ; 33), (64 ; 25), (50 ; 32), (54 ; 32)
On ponctue la liste des coordonnées par deux valeurs :

— 1000 pour indiquer que l'on doit passer au point suivant en levant le stylo

— 2000 pour indiquer que le dessin est terminé

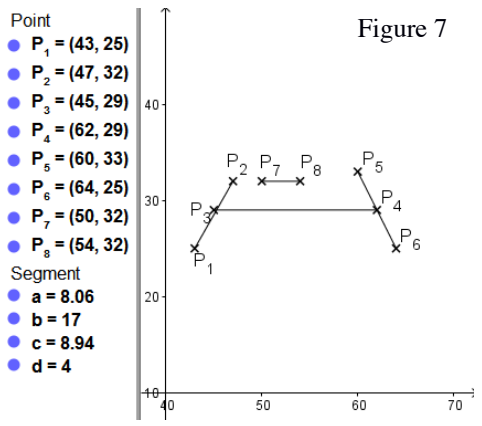


Figure 7

Ce qui donne :
43, 25, 47, 32, 1000, 45,29, 62, 29, 1000, 60, 33, 64, 25, 1000, 50, 32, 54,32, 2000
Sources (Delahaye, 1985, pp. 39-45)

II — Application

Vous devez écrire un programme qui, à partir des données qui vous seront fournies, permettra de faire dessiner ***n'importe quel*** visage.
Ci-contre, les premiers outils à votre disposition :

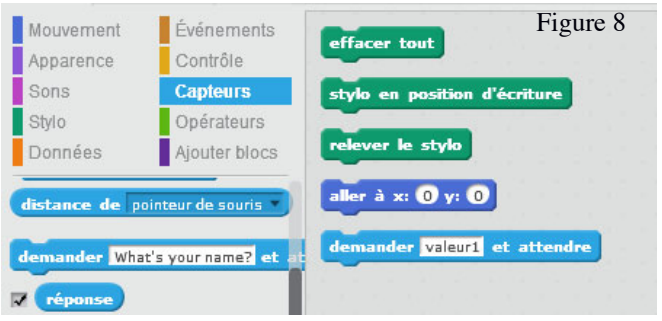
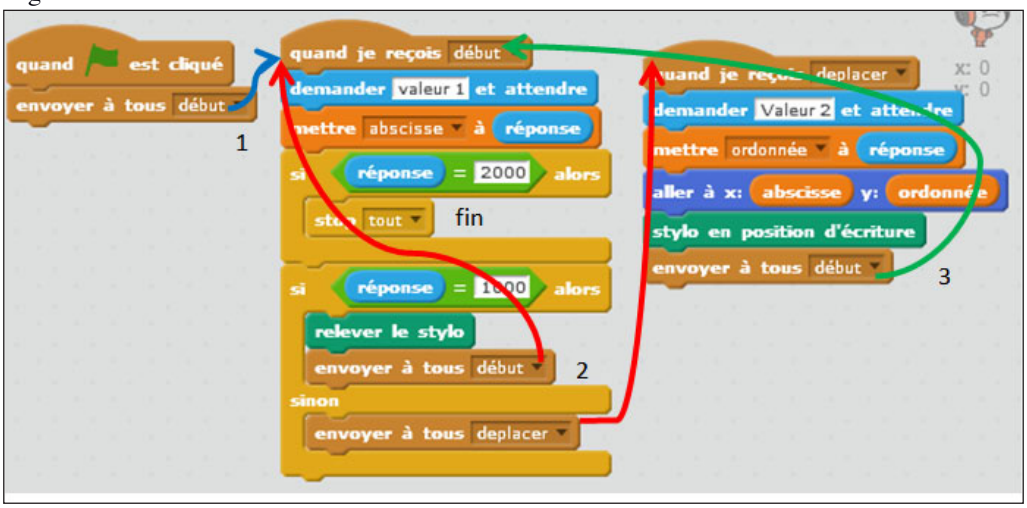


Figure 8

III — Transmettre au second lutin

1. Le second lutin doit recevoir les informations du premier lutin et produire le symétrique de la figure par rapport à l'axe des ordonnées.
2. Le second lutin se déplace ; son abscisse est l'opposé de celle du lutin 1 et son ordonnée est l'opposé de celle du lutin1. Que va produire le lutin 2 ?
3. Le second lutin se déplace et son abscisse est la somme de celle de lutin 1 et de – 100. Son ordonnée est la somme de celle de lutin 1 est de 30. Que va produire le lutin 2 ?
4. Le second lutin se déplace et son abscisse est le double de celle de lutin 1. Son ordonnée est le double de celle de lutin 1. Que va produire le lutin 2 ?

Figure 9



i.e. une tâche de planification des actions. Au sein de l'environnement Scratch, une implémentation possible dans le cadre de la programmation évènementielle est présentée, Figure 9, ci-dessus.

La figure 9 précise l'ordre dans lequel les instructions vont se dérouler. (1, puis 2 puis 3, avec des répétitions éventuelles pour 2 et 3). La gestion du flux étant en particulier assurée par les valeurs de la variable « *réponse* ».

Concernant la gestion du stylo, il s'agit d'identifier que ce dernier doit être abaissé après les déplacements du lutin. Il s'agit dès lors de s'assurer d'être dans le bon état (stylo baissé) après chaque déplacement du lutin, ceci même lorsque le stylo est déjà en position d'écriture pour ne pas avoir à rajouter une variable booléenne supplémentaire qui permettrait de connaitre l'état du stylo. Toutefois, la gestion de l'abaissement du stylo peut être laissée de côté dans un premier temps et reprise après avoir géré les déplacements du lutin, puisque des rétroactions graphiques (des segments non tracés ou des segments en trop) permettent d'identifier la place de cette instruction dans la séquence d'instruction à produire.

Le concept de *message* permet en outre une implémentation du problème proche du traitement à la main du problème, dans la mesure où il permet une forme de communication entre scripts et donc une forme de prise en compte d'une temporalité des actions à conduire dans la résolution du problème. Implémenter une telle solution nécessite ainsi de comprendre que l'instruction « *envoyer à tous début* » permet lors de son exécution de le relancer depuis le début. Il s'agit alors d'une forme de récursivité terminale. L'environnement Scratch crée l'événement associé au déclenchement de ce script. L'effet produit est donc le même que celui d'une boucle, la sortie de la boucle étant assurée par le test sur la variable *réponse*.

Eléments d'analyse du projet : les variables informatiques

Dans ce projet, trois types de variables sont à l'œuvre dans l'implémentation (Figure 10).

– *abscisse* et *ordonnée* sont des données explicites (variables exogènes) du problème. Dans la première partie du projet, elles seront mises à jour en n.2 et m.2 et *abscisse* pourra éventuellement se substituer à *réponse* dans les tests en n.3 et n.4. Elles serviront d'arguments pour la mise à jour des variables prédéfinies « *abscisse x* » et « *ordonnée y* » en m.3 et l.1 même si ces dernières ne sont pas visibles.
– *réponse* qui est une variable prédéfinie de l'environnement Scratch capture la saisie utilisateur (cette variable est à la fois endogène et prédéfinie). Elle servira dans les instructions d'entrées en n.1 et m.1, de mises à jour des variables *abscisse* et *ordonnée* en n.2 et m.2 et de test en n.3 et n.4.
– *stylo* du lutin 1 est une variable prédéfinie Scratch exogène qui ne peut avoir que deux états et dont il faut assurer l'initialisation en 0.1 et la mise à jour en n.5 et m.4 pour le lutin 1.
– *Stylo* du lutin 2 est une variable prédéfinie Scratch identique à stylo du lutin 1 mais indépendante.

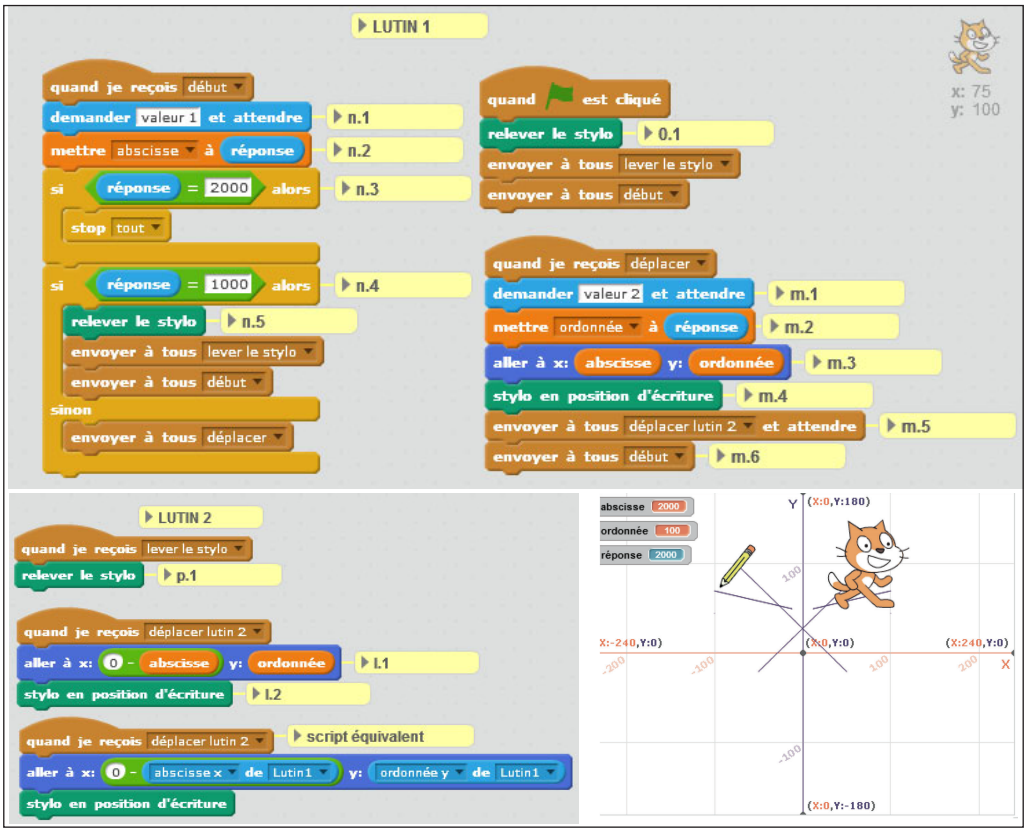


Figure 10

– « *abscisse x* » et « *ordonnée y* » de chacun des lutins sont des variables prédéfinies Scratch qui doivent être mises à jour. Leur état est visible à tout moment dans la fenêtre de script. L'opération de mise à jour de la position de lutin 1 en m.3 et celle de lutin 2 en l.1 font intervenir quelle que soit la question de la partie 3, l'expression algébrique d'une transformation des variables *abscisse* et *ordonnée* auxquelles peuvent se substituer les variables prédéfinies Scratch « *abscisse de lutin 1* » et « *ordonnée de lutin 1* » en l.1.

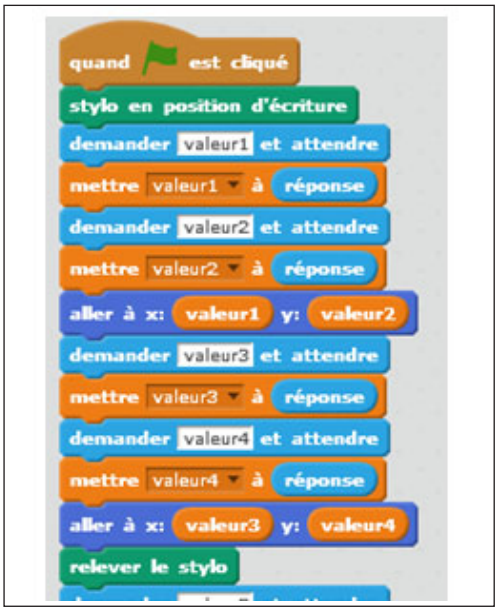
La gestion de la séquentialité/synchronisation des déplacements de lutin 1 et de lutin 2 relève de connaissances spécifiques de l'environnement Scratch que ce soit dans les opérations de mise à jour de la position des lutins ou pour capturer la saisie utilisateur. Cette instruction permet d'une part de déclencher l'évènement associé dans la boucle interne de Scratch et ainsi permettre l'exécution du script qui lui est associé mais aussi d'attendre la fin de l'exécution de ce jeu d'instructions pour poursuivre et exécuter l'instruction m.6. Ainsi au-delà des modifications à opérer pour effectuer les changements d'états nécessaires à la résolution du problème, ce projet engage les élèves dans un questionnement explicite sur l'état du système au prix de la construction de connaissances de l'environnement Scratch.

Analyse de productions d'élèves dans la réalisation du projet visage

Des élèves qui ne se détachent pas d'un jeu de données spécifique

MY (Figure 11) ne se détachent pas du jeu de données proposé dans la première partie de l'énoncé. Ils créent seize variables « *valeur1*, …, *valeur16* » qu'ils utilisent correctement, pour gérer

Figure 11



les affectations par saisie de l'utilisateur. L'automatisation de la prise en compte des marqueurs 1000 et 2000 est absente. Ils ne voient pas la nécessité de se détacher d'un jeu de données comme en témoigne la place des instructions permettant de relever le stylo.

Nous pouvons identifier deux origines à cette production. L'absence du concept de boucle chez les élèves ou la facilité qu'offre l'environnement Scratch quant à la duplication d'une partie de script. Le support proposé permet une rétroaction immédiate par l'enseignant désireux de tester un jeu de données qui ne correspond pas à la structuration présentée par MY.

Par ailleurs concernant l'usage des variables, les élèves utilisent autant de variables que de données. Là encore la taille des jeux de données étant différente d'un jeu à l'autre, un jeu de donnée différent permet de questionner cette pratique.

Figure 12

Toutefois, nous nous interrogeons sur le fait que l'usage d'une variable dont la mise à jour fait disparaitre des données, pourrait constituer un changement de contrat dans le cadre de la classe de mathématiques pour les élèves. Ce changement de contrat peut alors expliquer la réticence de ces élèves à ne créer que deux variables.

Pour finir, nous pouvons aussi questionner la compréhension de « n'importe quel », quantificateur présent dans la consigne et ainsi mettre en opposition la production de MY et celle de ET (Figure 12) qui pour leur part comprennent « n'importe quel » comme « en particulier celui que je veux/choisi » allant jusqu'à produire leur propre élément du visage. Il est alors difficile de démêler ce qui relève de la programmation et ce qui relève de la compréhension du quantificateur « n'importe quel ».

MS : *Une tentative de gestion en parallèle de la saisie utilisateur*

MS (Figure 13) essaient de gérer la double saisie de l'abscisse et de l'ordonnée d'un point au moyen de deux scripts qui s'exécutent en parallèle. Ils ne savent pas que dans Scratch, deux scripts déclenchés par le même évènement vont former une pile dans la partie interne du logiciel et qu'ensuite, une instruction de chaque script va être exécutée alternativement.

Leur procédure pourrait être correcte à condition de gérer le stylo pour ne pas voir tous les traits visibles à l'écran et de ne se déplacer que dans le second script utilisant

.

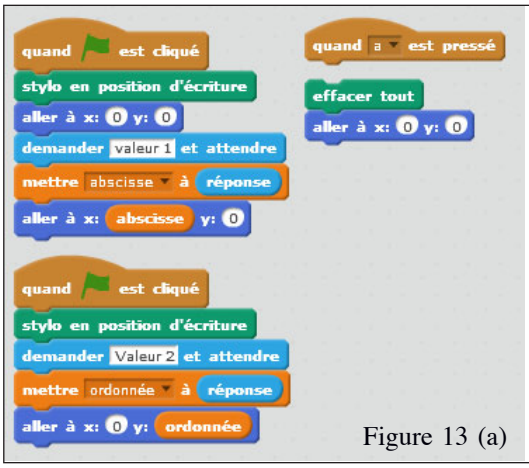


Figure 13 (a)

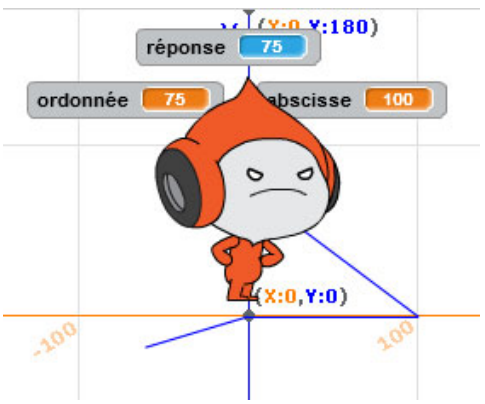


Figure 13 (b)

La gestion de la mise à jour des variables « *abscisse* » et « *ordonnée* » est correcte et le fait de voir ces deux informations correctes dans la scène (Figure 13 b.) déstabilise les élèves qui ne comprennent pas l'origine de leur erreur.

Nous pouvons supposer que les mises à jour des variables prédéfinies Scratch « *abscisse x* » et « *ordonnée y* » lors de l'exécution des instructions

sont transparentes pour les élèves.

Nous pouvons sans doute même aller jusqu'à une interprétation du rôle du zéro dans les champs correspondants qui « n'a pas d'effet » associé à des conceptions mathématiques erronées dans ce cas de figure : « neutre pour l'addition », « on ne l'écrit pas ».

Couplée à une mauvaise interprétation du rôle du zéro, cette production témoigne d'une conception erronée du parallélisme dans Scratch que l'on pourrait formuler ainsi : « puisque les deux scripts se déroulent simultanément, seules les modifications seront effectives, ainsi nous arriverons au point de coordonnées (abscisse ; ordonnée) ».

Le rendu graphique permet toutefois une possible avancée puisque visiblement, ce ne sont pas les opérations sur les variables qui n'ont pas été correctement implémentées mais les déplacements du lutin. Cette production montre que la représentation des élèves de l'environnement Scratch est en évolution.

ZEEH et la MNP Scratch

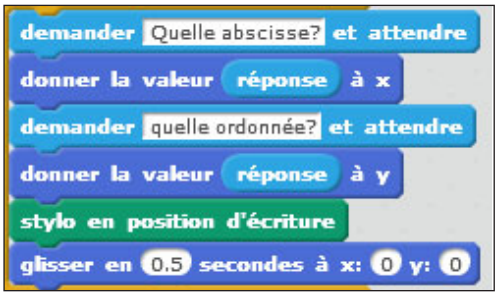
ZEEH (Figure 14) gèrent correctement les tests relatifs à la gestion des marqueurs 1000 et 2000. Dans un second temps, ils mémorisent les valeurs saisies par l'utilisateur dans la variable prédéfinie Scratch « *réponse* ». Cela implique que seule la dernière valeur saisie par l'utilisateur est conservée.

C'est sans doute pour cela qu'ils utilisent les instructions :

, pensant ainsi conserver les valeurs saisies directement dans les variables prédéfinies Scratch « *abscisse x* » et « *ordonnée y* ». Dans Scratch, ces deux instructions modifient dès leur exécution les variables prédéfinies Scratch « *abscisse x* » et « *ordonnée y* » du lutin auquel l'instruction est associée et déplace donc le lutin. Ils rajoutent une instruction de déplacement absolu à la suite, sans pour autant la renseigner (se retrouvant dès lors bloqués).

Ici nous pouvons nous interroger d'une part sur la sémantique des instructions, le verbe d'action « donner » ne réfère pas à un déplacement contrairement à « glisser » ou « aller ». Cette non correspondance sémantique entre la lecture de l'instruction et son effet dans Scratch semble alors source d'obstacle pour les élèves. D'autre part notre interrogation porte aussi sur le fait que les élèves ne semblent pas considérer pas les variables prédéfinies « *abscisse x* » et « *ordonnée y* » comme des variables puisque non définies/déclarées par le programmeur mais aussi peut être parce qu'elles

Figure 14



n'apparaissent pas ici dans des ovales comme les autres variables.

Demander « valeur1 »
et attendre : sémantique des blocs

Nous avons observé à plusieurs reprises, des épisodes témoignant de la difficile compréhension par les élèves des effets de certaines instructions dans la partie interne de l'environnement Scratch.

Les élèves ont produit l'assemblage de bloc 

pensant ainsi capturer la saisie de l'utilisateur et la mémoriser dans la variable « *valeur1* ». La non correspondance sémantique de l'instruction « demander valeur 1 et attendre » et son effet sur le système de variables dans Scratch est ainsi source d'obstacles.

L'instruction produite par l'élève permet de faire afficher le contenu de la variable « *valeur1* » dans une bulle attachée au lutin auquel cette instruction est associée. Elle aura pour effet d'interrompre tous les scripts pour attendre la validation de la saisie par l'utilisateur et ainsi affecter cette saisie à la variable prédéfinie Scratch « *réponse* ». Le champ que l'on remplit dans l'instruction 

à une forme rectangulaire ce qui signifie qu'il peut accueillir indifféremment une chaine de caractère ou un nombre.

La possibilité d'y intégrer un conteneur/une variable permet des interactions avec l'utilisateur qui peuvent ne pas être linéaires dans le sens où elles peuvent différer lors de l'exécution d'un même programme. La dénomination même de cette variable prédéfinie Scratch pose problème dans la mesure où à tout moment, l'élève doit se poser la question de savoir s'il parle de la variable prédéfinie Scratch « *réponse* » ou de la réponse qu'il vient de fournir lors de l'interaction.

Analyse de compte-rendus d'élèves
relatifs au traitement des donnés par le lutin 1

L'analyse de ces productions d'élèves permet de mettre en évidence la nécessité d'une prise en charge spécifique de certaines instructions relatives à Scratch qui peuvent, de part leur proximité sémantique avec le langage naturel, induire certains phénomènes qui vont se constituer comme autant d'obstacles à l'apprentissage de la programmation.

C'est ce que montre le phénomène 

qui cristallise un certain nombre de difficultés.

Nous mettons ce phénomène en perspective avec la volonté des concepteurs de Scratch de donner un sens à tout assemblage de blocs. Dans cette volonté, la correspondance sémantique de la « phrase produite » et de son effet au sein de l'environnement n'est pas toujours assurée. Ceci demande des connaissances qui ne sont pas seulement de l'ordre de l'appropriation des instructions mais qui se réfèrent à la compréhension du fonctionnement de l'environnement.

Dans son compte-rendu (Figure 15 de la page suivante), UL écrit que abscisse et ordonnée sont les réponses aux questions 1 et 2. L'explication qu'il fournit de l'instruction « mettre ordo à réponse » démontre une confusion possible : « mettre ordo à réponse pour grader ***la réponse*** en mémoire ». Il y a ici confusion entre la réponse fournie et le contenu de la variable *réponse*.

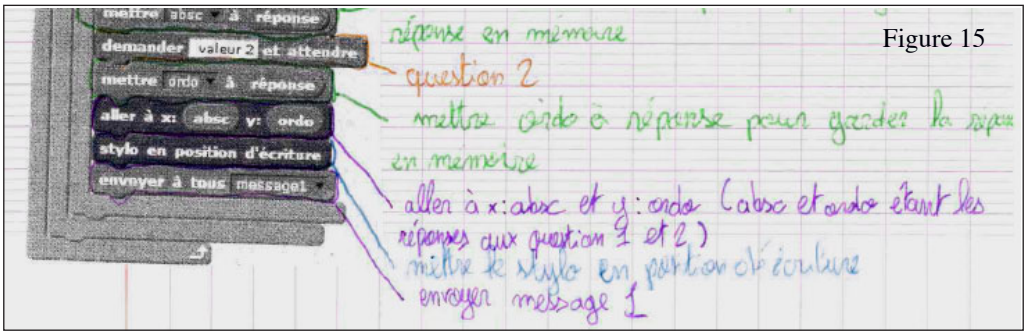


Figure 15

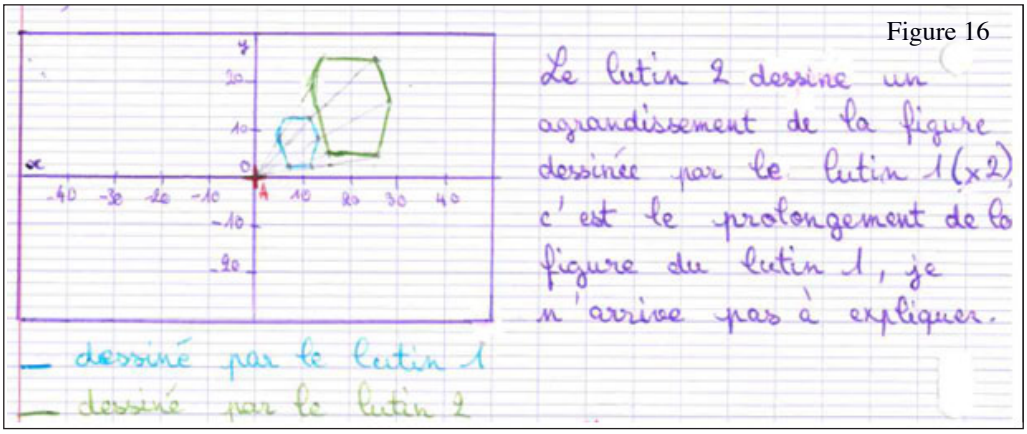


Figure 16

Analyse des productions des élèves relatives à l'expression des transformations : Variable mathématique et variable informatique toutes deux variables mais distinctes

Dans la production de SM (Figure 16), des difficultés à revenir au cadre mathématique pour justifier la nature des transformations existent. La variable utilisée dans la cadre de la programmation ne peut être la même que celle utilisée en mathématiques. Ce qui tend à souligner la différence de perception et de traitement entre les variables mathématiques et informatiques. Pour autant, SM (Figure 17) sont capables de produire une formule correcte pour programmer le lutin lightning, elles identifient le rôle des variables « *abscisse* » et « *ordonnée* » pour produire une opération arithmétique servant à la mise à jour des variables prédéfinies Scratch « *abscisse x* » et « *ordonnée y* » du lutin lightning.

Figure 17

Dans la production d'une solution informatique en réponse à un problème, ce sont

les changements d'états des variables qui permettent d'aboutir à la solution. Cette conceptualisation de la variable informatique ne coïncide pas avec la conceptualisation de la variable en mathématiques. Les élèves réalisent une figure et identifient une situation d'agrandissement de coefficient 2. Elles ne codent pas la figure produite et ne sont pas capables de produire une explication. Ce sont ici des connaissances algébriques relatives à la lettre en tant que nombre généralisé qui sont absentes ou en décalage avec celles requises sur la variable informatique. Nous pouvons supposer plusieurs origines à ces difficultés :

– les élèves ne s'autorisent pas à utiliser les variables « *abscisse* » et « *ordonnée* » pour coder la figure produite (Figure 16) du fait du changement de statut de ces variables/lettres nécessaire (en informatique, on focalise sur l'état et la transformation, en mathématiques, on généralise).
– les élèves ne négocient pas le changement de point de vue de global à local dans le sens où elles observent l'effet de la transformation sur la figure mais ne prennent pas un point (n'importe lequel) qui permettrait de faire le lien avec l'effet du script de la (Figure 17).

Nous pouvons opposer la production ci-dessus à la production de TF Figure 18 qui comprend parfaitement le rôle de la lettre en mathématiques et le rôle de la variable en informatique et distingue les deux objets allant jusqu'à les nommer différemment. Il se trompe entre abscisse et ordonnée (uniquement dans son écrit) mais TF distingue la lettre *x* qui représente un nombre généralisé et la variable informatique « *ord* ».

Diverses expressions produites pour construire l'opposé

Majoritairement les élèves ont produit des expressions pour construire l'opposé qui ne laissent pas de champs vides. Très peu ont directement utilisé 

. Certains sont même allés jusqu'à produire 

expression qui convoque des connaissances sur l'usage des opérateurs binaires dans l'environnement Scratch relativement à la gestion des priorités opératoires. Ici de par l'emboitement des opérateurs, l'élève a produit l'expression ((abs-abs)-abs). Chaque opérateur binaire étant équivalent à une paire de parenthèses.

La complexité des solutions produites peut être expliquée par le fait que les opérateurs

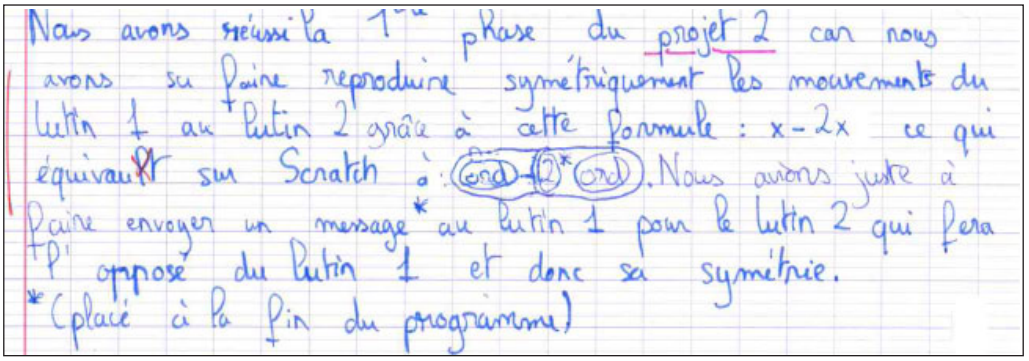

Nous avons réussi la 1ère phase du projet 2 car nous avons su faire reproduire symétriquement les mouvements du lutin 1 au lutin 2 grâce à cette formule : x - 2x ce qui équivaut sur Scratch à : ord - (2 * ord). Nous avons juste à faire envoyer un message* au lutin 1 pour le lutin 2 qui fera l'opposé du lutin 1 et donc sa symétrie.
*(placé à la fin du programme)

Figure 18

binaires de Scratch sont vides  . Il s'agit pour l'élève de remplir les deux champs. Les élèves ne semblent pas s'autoriser à remplir ces opérateurs de manière mixte, par une expression et un nombre (« 0 ») (encore une fois peut-être du fait du statut particulier de « 0 »).

La variété des expressions produites pour construire l'opposé est liée aux spécificités du langage Scratch en termes d'écriture d'expressions algébriques. Il y a une différence de traitement entre l'écriture mathématique et l'écriture informatique dans l'environnement Scratch d'une expression algébrique. Le fait d'être toujours obligé d'utiliser un opérateur binaire dans lequel il faut remplir deux champs est un geste différent de celui qui s'opère en mathématique où il s'agit d'identifier la structure de l'expression algébrique. La construction d'une expression algébrique dans le langage Scratch qui relève de la connaissance des priorités opératoires et de la structure de l'expression algébrique produite est proche de l'arbre algébrique qui permet d'expliciter la structure de l'expression. Les opérateurs binaires en Scratch sont équivalents à des parenthèses et modifie le regard porté sur les expressions algébriques. Cela constitue un changement par rapport aux mathématiques où par exemple, il n'est pas nécessaire d'expliciter l'opération pour écrire $-x$, l'opposé de x.

Conclusion

Dans cet article nous avons examiné la construction des connaissances informatiques dans un contexte de classe de mathématiques au cycle 4, en prise d'appui sur l'environnement Scratch. En raison de son importance en informatique et en mathématiques, l'étude s'est centrée en particulier sur le concept de variable. Il s'est agi de donner à voir les multiples facettes de la variable informatique à l'œuvre dans des activités de programmation au sein de l'environnement Scratch (valeur temporaire pour l'échange de valeurs entre 2 variables ; initialisation, mise à jour et vérification / test dans le cadre d'une itération / boucle ; variable locale vs. variable globale ; "message" pour activer des événements ; implication dans le flux de contrôle entre les appels des scripts ; etc.). Il s'est agi également d'examiner les phénomènes à l'œuvre dans la conceptualisation de ces différentes acceptions de la variable.

Ce faisant, nous avons dévoilé une forme de complexité attenante à la conceptualisation de la variable informatique, complexité par ailleurs souvent masquée au sein de l'environnement Scratch, eu égard à ses spécificités qui rendent muette une discussion sur la notion de typage des variables. Par ailleurs, nous avons examiné comment les concepts de variable informatique et de variable mathématique s'influencent réciproquement (modélisation de la variable en tant qu'une seule valeur vs. modélisation en tant qu'une série des valeurs ; la lettre en algèbre, en tant que nombre généralisé vs. la variable informatique ; construction de l'opposé ; etc.).

L'ensemble de ces réflexions concernent plus largement la question de la représentation (des données et/ou du traitement) informatique d'un problème sur les « objets du monde », pour arriver à la production d'un texte de programme exécutable sur un dispositif informatique. Dans les différentes voies susceptibles d'être empruntées dans cette activité de programmation, les connaissances potentiellement en jeu diffèrent et convoquent des usages de la variable informatique et mathématiques distincts. L'environnement Scratch ouvre la possibilité de construire différentes facettes de la variable informatique. Dans le même temps, pour cer-

taines de ces facettes, une attention particulière est nécessaire, parce que moins visibles ou plus complexes dans cet environnement logiciel ("message", lien avec le flux de contrôle pour l'itération et les appels des scripts). Il n'en reste pas moins que des activités menées en classe dans lesquelles sont en jeux une pluralité de rôles, d'usages, de types de variables sont susceptibles de co-construire de connaissances mathématiques et informatique sur la variable.

Bibliographie

Baron, G.-L., Bruillard, E., & Komis, V. (2011). In G.-L. Baron, E. Bruillard & V. Komis (dir.), Sciences et technologies de l'information et de la communication (STIC) en milieu éducatif ; analyse des pratiques en enjeux didactiques. Athènes, Université de Patras.

Delahaye, J-P (1985). Dessins géométriques et artistiques avec votre micro-ordinateur, Eyrolles, pp 39-45.

Dowek, G. (2005). Quelle informatique enseigner au lycée ? http://www-roc.inria.fr/who/Gilles. Dowek/lycee.html

Hoc, J.-M. (1977). Méthode d'analyse psychologique d'un travail de programmation. Le Travail Humain (40), pp. 15-28.

Hoc, J.-M. (1984). Les activités de résolution de problème dans la programmation. Psychologie Française, 29 , pp. 267-271.

Nivat, M. (2009). Vaches et informatique. In G.-L, Baron, E, Bruillard & V., Pochon (dir.), Informatique et progiciels en éducation et en formation : continuités et perspectives (p. 28-38). Lyon. France

Rogalski, J. (1988, Septembre). Enseignement de méthodes de programmation dans l'initiation à l'informatique. Colloque francophone sur la didactique de l'informatique, pp. pp.61-72.

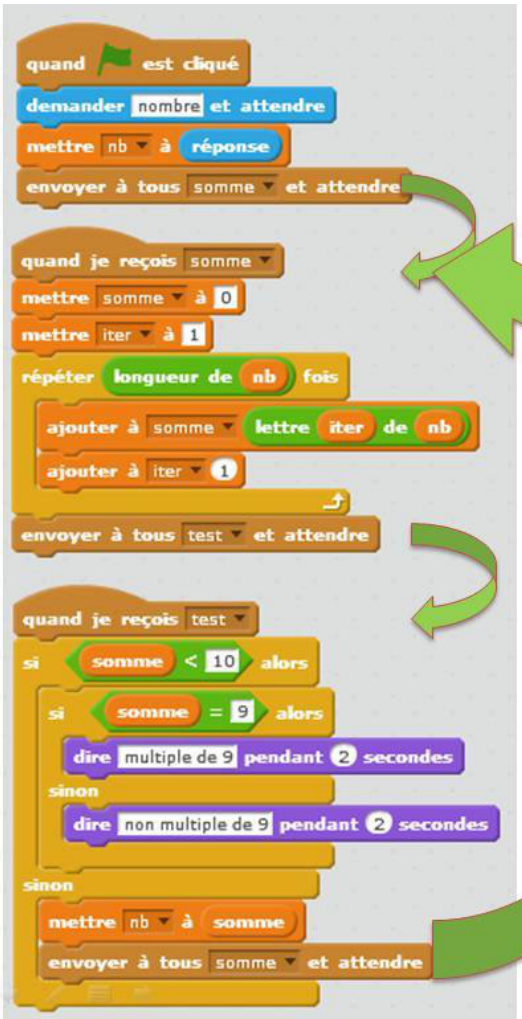
Samurçay. (1985). Signification et fonctionnement du concept de variable informatique chez les élèves débutants. 16, p. 144.

Samurçay, R. (1988). Modèles cognitifs dans l'acquisition des concepts informatiques. Actes du premier colloque franco-allemend de didactique des mathématiques et de l'informatique , p. 219.

Samurçay, R., & Rouchier, A. (1985). De «faire» à «faire faire» : planification d'actions dans la situation de programmation. Enfance , 2-3, pp. 241-254.

Viallet, F., & Venturini, P. (2010). Didactique comparée et enseignement de l'informatique. In Actes du congrès de l'Actualité de la recherche en éducation et en formation (AREF). Genève.

ANNEXE


quand est cliqué
demander nombre et attendre
mettre nb à réponse
envoyer à tous somme et attendre
quand je reçois somme
mettre somme à 0
mettre iter à 1
répéter longueur de nb fois
ajouter à somme lettre iter de nb
ajouter à iter 1
envoyer à tous test et attendre
quand je reçois test
si somme < 10 alors
si somme = 9 alors
dire multiple de 9 pendant 2 secondes
sinon
dire non multiple de 9 pendant 2 secondes
sinon
mettre nb à somme
envoyer à tous somme et attendre