SAVOIRS, CONCEPTS ET SITUATIONS DANS LES PREMIERS APPRENTISSAGES EN PROGRAMMATION ET EN ALGORITHMIQUE

Jean-baptiste LAGRANGE et Janine ROGALSKI
LDAR, Université Paris-Diderot
jb.lagrange@casyopee.eu
rogalski.muret@gmail.com

INTRODUCTION

L'enseignement de l'algorithmique et de la programmation est revenu dans le curriculum récent sans référence à l'expérience acquise antérieurement, sans exploitation du questionnement laissé ouvert après des tentatives antérieures et sans tenir compte de l'existence de recherches menées dans des contextes divers, scolaires ou non. Cet article vise donc à rétablir les liens manquants de façon à ce que des recherches puissent se développer sans se priver de l'acquis de travaux menés depuis plus de trente ans sur les apprentissages en programmation et en algorithmique. Il est écrit par deux chercheurs travaillant dans le même laboratoire, mais appartenant à des champs différents, la didactique des mathématiques et la psychologie ergonomique. Dans de nombreux domaines de l'enseignement et de l'apprentissage, un double point de vue impliquant ces deux champs s'est montré pertinent et fructueux. Nous considérons ici les élèves et leurs apprentissages en programmation et en algorithmique tels que prescrits dans un contexte scolaire que nous allons préciser. La didactique des mathématiques dispose d'outils et de méthodes pour interroger la réalité de ces apprentissages : comment les élèves apprennent-ils, quelles difficultés rencontrent-ils, quels sont réellement les savoirs en jeu ? Au-delà de ce contexte et de cette prescription, les apprentissages concernent un domaine, l'informatique, à la fois champ scientifique et domaine professionnel. La psychologie ergonomique comprend un champ travaillant à la "psychologie de la programmation" et collabore avec une "didactique de l'informatique" en devenir. L'activité de programmation y est considérée au sens large incluant les différents aspects du développement logiciel et les apprentissages sont analysés dans la transposition de pratiques de la programmation et de savoirs informatiques. Ceci concerne les élèves que nous considérons ici en tant qu'"apprentis programmeurs" et l'enseignement qu'ils reçoivent en tant qu'"alphabétisation informatique". L'existence pérenne de la psychologie de la programmation depuis bientôt 30 ans rend incontournable la prise en compte de ses acquis dans toute recherche y compris sur des débutants.

Après avoir brièvement rappelé en introduction dans quelles conditions des apprentissages en programmation et en algorithmique ont pu exister antérieurement dans le curriculum scolaire, notre approche dans cet article consiste dans une première partie à mettre en correspondance certaines observations dans le contexte curriculaire récent, avec des phénomènes analysés dans un contexte plus ancien, de façon à assurer la continuité du questionnement sur les apprentissages en programmation et en algorithmique, par-delà les différences de contexte. Nous recherchons aussi dans une seconde partie comment la question des situations d'apprentissage s'est posée d'un contexte à l'autre. Ces deux premières parties

s'appuient principalement sur l'expérience du premier auteur, avec un point de vue marqué par la didactique des Mathématiques. Le choix y est fait de conserver une position ouverte relativement aux savoirs en jeu : ces savoirs sont très peu spécifiés par le curriculum, et, nous allons le montrer, n'ont pas d'existence claire ni dans les mathématiques scolaires ni dans l'informatique en tant que domaine scientifique ou professionnel. Il s'agit de mobiliser les outils de la didactique comme science expérimentale pour mieux cerner ces savoirs et les situations qui y correspondent. La troisième partie répond à la nécessité d'élargir le propos au-delà des expériences isolées qui alimentent les deux premières parties. Elle s'appuie particulièrement sur l'expertise du second auteur pour proposer un bilan des acquis de la psychologie de la programmation, précédé par une présentation de ses acteurs et de son évolution. Ce bilan se particularise sur un champ conceptuel précisément identifié, celui de "variable informatique". Pour conclure, nous joignons les deux points de vue pour souligner les questions qui nous semblent prioritaires.

D'un contexte à l'autre

Au début des années 1980, l'ordinateur devient un objet assez accessible, mais sur les machines grand public ou pour l'enseignement, les usages possibles sont surtout limités à la programmation. Le livre d'Arthur Engel "Elementarmathematik vom algorithmischen Standpunkt " parait en 1977 et est traduit en français au début des années 1980 (Engel, 1980). Il propose que les mathématiques scolaires mettent l'accent sur les algorithmes, leur construction et leur mise à l'épreuve plutôt que sur leur exécution. Le livre de Papert, "Mindstorm" sur les apports de la programmation Logo paraît aussi en français à la même époque (Papert, 1981). Ces ouvrages témoignent de ce que les activités de programmation sont vues par beaucoup d'innovateurs, comme pouvant contribuer aux apprentissages dans les domaines scientifiques existants, les mathématiques en premier lieu. Cela se traduit par quelques pratiques et groupes de recherche dans les IREM, sans toutefois qu'il y ait un impact sur le curriculum, au moins en France. En revanche, l'institution met en place un enseignement centré sur l'informatique comme champ scientifique, l'option informatique des lycées. Cette option se généralise très vite sous la pression des familles. Il s'agit d'offrir un nouveau champ scientifique, tout en soulignant les liens qu'il entretient avec les champs scolaires existants. Le volume horaire est de 2,5 h par semaine, avec le thème central des méthodes de construction de programmes. Ainsi, l'option informatique présente une forte dimension programmation (Baron, 1989).

Au tournant des années 1990, les activités de programmation disparaissent brusquement du curriculum du lycée, avec l'arrêt de l'option informatique et à d'autres niveaux scolaires les quelques pratiques existantes, surtout autour de Logo, s'arrêtent aussi. Les raisons sont complexes. Une raison peut cependant être soulignée : c'est l'irruption des progiciels comme le tableur, et aussi des applications dédiées comme le calcul formel et la géométrie dynamique qui donnent l'impression que l'enseignement tirera plus facilement profit de ces nouvelles applications que des activités de programmation. Ainsi Ruthven (1996) écrit:

> Building expertise and fluency in programming requires a considerable investment of time and effort, and imposes significant intellectual demands in its own right. In purely instrumental terms, newer tools, such as the spreadsheet, offer comparable power and versatility with lower overheads of technique and greater interactive flexibility. (op.cité, p.458)

La résurgence actuelle des activités de programmation peut aussi s'interpréter comme une prise de conscience, 20 ans après, de ce que d'une part les activités avec des progiciels ou des applications dédiées, ne sont pas non plus si faciles à mettre en œuvre, et de ce que, d'autre part, elles ne conduisent pas nécessairement à la compréhension des principes et méthodes qui

sous-tendent leur conception.

OBSERVATIONS ET PHENOMENES DANS LE CONTEXTE ACTUEL ET DANS L'OPTION INFORMATIQUE

Nous retenons que l'expérience des années 1980 telle qu'évoquée par Ruthven, contredit l'idée d'un accès aisé des élèves à une expertise suffisante pour que les activités en programmation et algorithmique puissent contribuer à des apprentissages. Nous recherchons, à travers des exemples pris dans des travaux de recherche, comment des questions liées la construction de cette expertise se posent dans les deux contextes, et comment elles renvoient à des savoirs et aux situations qu'il est nécessaire de mettre en place pour que les élèves se confrontent à ces savoirs ou concepts. Nous considérons dans cette première partie deux exemples de difficultés résistantes l'une ayant trait de façon générale aux spécificités d'un langage informatique et l'autre portant plus particulièrement sur le statut donné aux "conditions" dans le langage informatique

Les spécificités d'un langage informatique

Nous avons repéré cet exemple dans une thèse récente (Briant, 2013). L'auteure note le changement institutionnel que constitue l'introduction, dans toutes les classes de la voie générale du lycée, de l'algorithmique en mathématiques. Parmi les questions abordées dans la thèse, on trouve la contribution de ce nouvel enseignement à la compréhension des concepts mathématiques et plus particulièrement l'apport de l'algorithmique à l'apprentissage de concepts algébriques, ce qui rejoint un thème ayant déjà fait l'objet de recherches dans les années 1980, notamment par Tall et Thomas (1986). Elle présente une situation dont l'objectif est de montrer aux élèves que la résolution de certaines équations peut se faire de manière algorithmisée, et ainsi de les amener à comprendre que la reconnaissance de la nature et de la forme d'une équation leur permet de choisir dans" leur catalogue cognitif de techniques". Une première tâche, que nous prenons en exemple, consiste pour les élèves à réaliser un algorithme prenant en entrée trois réels a, b et c, et donnant en sortie la solution quand elle existe et est unique de l'équation ax+b=c, ou un message adéquat dans les autres cas. Briant (2013) donne une variété de solutions d'élèves dont nous extrayons certaines dans le tableau 1.
La solution attendue est proche de la solution "élève" n°1 du tableau 1. Pour que cet algorithme soit complet, il faudrait que l'élève traite par deux alternatives les différents cas où n'existe pas une solution unique (en l'état, l'algorithme est en erreur dans le cas a=0). Certains élèves produisent cette solution. Elle témoigne d'une prise de conscience de la capacité du dispositif à traiter une formule écrite dans une syntaxe très proche de l'algèbre habituel, en instanciant les trois paramètres selon les valeurs données en entrée. Il est attendu que les élèves rencontrent la particularité du cas a=0 lors de l'exécution, ce qui les conduira à considérer la résolution de cette équation de façon plus générale. D'autres élèves produisent des solutions du type de celles numérotées 2 et 3 dans le tableau. Elles témoignent d'une incompréhension du langage informatique, de la séquencialité et des variables. Les élèves essaient simplement de traduire l'équation avec les éléments syntaxiques du langage, soit par une affectation (Solution "élève" n°2), soit par une condition (Solution "élève" n°3). Les solutions du type de celle numérotée 4 dans le tableau sont aussi très souvent rencontrées : la variable I prend les valeurs successives du second membre lors d'une résolution en papier/crayon. Comme le note l'auteure, l'élève code les actions successives lors de cette résolution. Le programme est syntaxiquement correct et donne bien la valeur attendue, mais reste distant de de la solution attendue.

<table>
<tr>
<td>Solution"élève" n°1
DEBUT_ALGORITHME
LIRE a
LIRE b
LIRE c
x PREND_LA_VALEUR (c-b)/a
AFFICHER x
FIN_ALGORITHME</td>
<td>Solution"élève" n°3
DEBUT_ALGORITHME
SI (ax+b=c) ALORS
DEBUT_SI
FIN_SI
FIN_ALGORITHME</td>
</tr>
<tr>
<td>Solution"élève" n°2
DEBUT_ALGORITHME
LIRE a
LIRE b
c PREND_LA_VALEUR aX+b
FIN_ALGORITHME</td>
<td>Solution"élève" n°4
DEBUT_ALGORITHME
LIRE a
LIRE b
LIRE I
I PREND_LA_VALEUR I-b
I PREND_LA_VALEUR I/a
x PREND_LA_VALEUR I
AFFICHER x
FIN_ALGORITHME</td>
</tr>
</table>

Tableau 1: quatre solutions pour une équation du 1er degré. D'après Briant (2013 p. 414)

Voici comment des difficultés de même type ont été observées et analysées dans le contexte de l'option informatique (Lagrange, 1991). Il s'agit d'une séquence d'apprentissage sur le type chaine de caractères en classe de Seconde, dont l'objectif général était la prise de conscience par les élèves du caractère calculable des objets du langage correspondant à ce type. Cette séquence était conçue comme une remédiation, les élèves ayant déjà eu une quarantaine d'heure en option et ayant été initiés au type chaîne de caractères Nous analysons ici une tâche (tableau 2) servant à une évaluation diagnostique en préalable à la séquence.

<table>
<tr>
<td>Enoncé
On donne le programme
<code>Lire X</code>
<code>A←souschaine(X,1,1)</code>
<code>L←longueur(X)</code>
<code>B←souschaine(X,L,1)</code>
<code>afficher A;B</code>
1. Indique ce que l'ordinateur affiche pour l'entrée de "BONJOUR", de"B"</td>
<td>2. Ecris un programme pour que l'utilisateur ayant entré une chaîne, l'ordinateur affiche la chaine en passant la première lettre à la fin
(BONJOUR → ONJOURB)
3. Ecris un programme pour que l'utilisateur ayant entré une chaîne, l'ordinateur intervertit la première lettre et la dernière
(TALUS→ SALUT)

Réponse d'élève aux questions 2 et 3
2. <code>A←souschaine(X,2,L+souschaine(1,1))</code>

3. <code>A←souschaine(X,2,L+souschaine(1,1))</code>
<code>A←souschaine(X,1,L-1+souschaine(5,1))</code></td>
</tr>
</table>

Tableau 2 : une tâche sur les chaînes; l'énoncé et une réponse d'élève aux questions 2 et 3

Dans le texte de la tâche (Tableau2), les deux fonctions principales, `longueur` et `souschaine` étaient d'abord rappelées. Dans une première question, il s'agissait d'interpréter un petit programme. Cela donnait aussi une sorte de modèle pour la suite. Dans les deux autres questions, il était demandé de créer un programme de même nature ; le langage était volontairement proche d'un traitement "à la main" (passer, intervertir…) plutôt que d'un calcul. Pour la première question, les résultats avaient été donnés correctement sauf par un élève qui donnait B au lieu de BB pour l'entrée de la lettre B. Pour la seconde et la troisième question, environ la moitié des élèves ne donnaient pas une réponse qui représentait le calcul d'une chaîne. Le tableau 2 donne une réponse typique d'élève aux questions 2 et 3. On voit qu'il y a bizarrement l'addition d'une variable représentant probablement la longueur (mais non

affectée) et d'un appel à `souschaine` incorrect puisqu'avec deux arguments seulement. La première ligne de la réponse à la question 3 reprend la réponse à la question 2 : l'élève a d'abord passé le premier caractère en dernier. La seconde ligne correspond au traitement du dernier caractère de la chaine initiale. On remarque que l'élève affecte le résultat à la même variable dans les deux lignes.

L'épreuve a été suivie d'un entretien devant l'ordinateur avec certains élèves ayant donné ce type de réponse. Nous analysons un entretien typique. Dans cet entretien, l'observateur a d'abord tenté une correction syntaxique, soulignant notamment l'absence d'affectation à L, l'addition d'un nombre et d'une chaîne et l'absence d'une instruction de sortie du résultat. Il a constaté que les corrections apportées ne faisaient pas sens pour l'élève. Revenant à la réponse de l'élève à l'épreuve, il a demandé ce que cette réponse signifiait pour elle. Voici une partie de l'entretien entre l'observateur (O) et l'élève (E).

O. Tu avais mis un"+" ici, que signifie-t-il ?

E. Ben, là… j'avais écrit `BONJOUR` donc `X,` à partir de la deuxième, on prend la longueur totale, plus la première lettre, on en prend une.

O. C'est quoi `souschaine(1,1)` …?

E. La longueur de `X` et `1` et `1`

Les réponses montrent que cette élève se servait des fonctions sur les chaînes pour coder des actions et non pour faire des calculs. `L+souschaine(1,1)` veut dire :"on prend la longueur totale, plus la première lettre". L'affectation n'a pas de signification, elle est là simplement pour la syntaxe, c'est pourquoi elle est répétée sur la même variable. Pour l'élève, `souschaine(1,1)` veut dire "la longueur de X et 1 et 1". On est bien dans une action sur un objet concret qu'il n'est pas nécessaire de renommer. On remarque dans la seconde ligne de la question 3 que l'élève opère sur `X`, comme si cette chaine avait été transformée par la première, puisqu'elle commence à 1 et s'arrête à `L-1`. Les remédiations opérées auprès d'élèves dans la suite de la séquence ont consisté en une série d'exercices du même type, traités en entretien devant l'ordinateur, l'intervenant essayant de comprendre avant de corriger. Elles ont été relativement payantes puisque dans une épreuve à la fin de l'année, ces élèves très en difficulté obtenaient des résultats semblables au reste de la classe. Cette recherche et d'autres que nous étudierons en seconde partie, amènent à penser que cette absence de prise de conscience des spécificités d'un langage informatique, est une difficulté importante sur la voie de l'expertise nécessaire pour que les objectifs curriculaires relatifs aux activités en programmation et en algorithmique soient atteints.

Le statut donné aux "conditions" dans le langage informatique

Dans un second exemple pris dans le contexte actuel de l'algorithmique au lycée, nous allons montrer comment l'écriture d'alternatives pose la question du statut donné aux "conditions" par les élèves et comment une enseignante bien préparée peut se trouver désarçonnée par les difficultés que cela entraîne pour la gestion de la situation. Puis nous irons plus loin dans l'analyse de cette difficulté et de son traitement en reprenant ici aussi une recherche menée dans le cadre de l'option informatique.

L'extrait suivant est tiré d'un épisode observé dans le cadre d'un mémoire de master (Guy, 2013) mettant en place une ingénierie sur laquelle nous reviendrons en seconde partie de cet article. Il s'agit d'une phase de mise en commun après la construction d'un traitement itératif devant se terminer quand une variable A prend une des deux valeurs 495 ou zéro. Il s'agit d'une bonne classe de terminale qui fait de l'algorithmique depuis la seconde. L'auteure du mémoire qui a préparé l'ingénierie la met en place en prenant la place de l'enseignant (P dans le dialogue) Le professeur de la classe est là en tant qu'observateur (O). Le langage choisi impose une

itération Tant que, c'est-à-dire avec une condition de continuation en début de boucle. L'extrait est une discussion collective qui fait suite à ce qu'un groupe (« les filles devant") a écrit la condition de continuation sous la forme (A différent de 495) ou (A différent de 0) et ne comprennent pas pourquoi l'exécution ne termine pas.

> P : ..vous les filles devant, vous m'avez dit que le test qu'on fait là c'est A différent de 495 **ou** A différent de 0, au fond, tu m'as dit. . .
>
> E3 : A différent de 0 **et** A différent de 495.
>
> P : Alors, qu'est-ce qui est correct, qu'est-ce qui ne l'est pas ? … Oui ?
>
> E7 : Ben moi ça me parait bizarre qu'un nombre soit égal à deux nombres différents.
>
> P : Voilà, donc, quand tu dis le et, tu vois, un nombre qui est différent de deux nombres, c'est vrai, tu vas trouver des nombres qui sont différents de 495 et différents de 0. .. Mais tu vas jamais satisfaire la condition être égal à 495 et à 0. … Finalement, ton algorithme, il s'arrêtera quand tu obtiendras un nombre égal à 495 et à 0, d'accord ? Et ça tu vas pas pouvoir le trouver.
>
> E3 : On l'a fait fonctionner et ça a marché.
>
> P : Tu l'as fait tourner et ça marche. . .
>
> O : Tu l'as fait. . . Ça devrait pas marcher.
>
> P : Oui, mais, AlgoBox il a des conditions d'évaluation des. . .
>
> …
>
> P : Bon, on va y réfléchir, c'est une question à laquelle il faut réfléchir …

L'enseignante P essaie de provoquer un débat. Un autre élève E3 corrige sans plus d'explication. Une élève E7 fait une intervention peu explicite, mais on imagine qu'elle pense à ce qui ferait sortir de la boucle, c'est-à-dire la négation de la condition (A<>495) et (A<>0), qu'elle conçoit comme (A=495) et (A=0),
L'enseignante reprend cette idée, mais de façon assez confuse, puisqu'elle considère en même temps une condition de continuation correcte "des nombres qui sont différents de 495 et différents de 0" et une condition d'arrêt fausse "être égal à 495 et à 0". La discussion tourne court, quand l'élève E3 qui a corrigé propose une validation pragmatique, reprise par le professeur. Le débat est relancé par l'observateur qui a dû être embrouillé par l'explication car il semble très étonné que la condition avec le "et" fonctionne. Mais la discussion reste sur le plan pragmatique, l'enseignante faisant référence au logiciel et non plus à la logique, et, après une phase assez confuse, clôt le débat. Dans son mémoire, l'enseignante dit qu'elle ne s'était pas préparée à ce problème. Pourquoi une enseignante bien formée, intéressée par l'algorithmique puisqu'elle fait son mémoire sur ce sujet est-elle ainsi prise au dépourvu ? Pourquoi ne reprend-elle pas la remarque de E7, en l'exprimant en termes de condition d'arrêt, et en soulignant que la condition dans le tant que est une condition de continuation, négation de la condition d'arrêt ? Ceci suggère que les difficultés liées aux conditions (valeurs logiques de type booléen) sont sous-estimées. La ressemblance entre ces conditions et celles que l'on exprime dans le langage "ordinaire" est trompeuse dès que les conditions deviennent un peu complexes, notamment lorsqu'elles imposent l'emploi de connecteurs.

Comme annoncé, nous précisons cette hypothèse en revenant sur une recherche menée dans le cadre de l'option informatique. Lagrange (1991) présente une ingénierie sur le type booléen expérimentée avec des élèves dans leur deuxième année d'option dans une classe scientifique après que les élèves aient été initiés à ce type de données. Cette ingénierie reposait sur deux problèmes. Chaque problème a donné lieu à une série de tâches en papier crayon et sur ordinateur que nous ne détaillons pas ici. Le tableau3 donne l'énoncé du premier problème,

une solution attendue et une solution "élève" typique. L'énoncé est conçu pour confronter les élèves aux spécificités des booléens. Une première spécificité est que l'entrée ou la sortie d'une valeur logique ne peut se faire directement, elle doit se faire via une chaîne de caractère. Une seconde spécificité est que les calculs sur ce type se font via des connecteurs logiques (conjonction, disjonction…) dont la similarité avec les expressions du langage courant est trompeuse, comme nous l'avons vu avec l'exemple précédent. Ainsi, l'énoncé présente une rupture entre d'une part la première et la seconde condition qui peuvent s'exprimer sous une forme proche du langage courant, et d'autre part la troisième qui implique d'articuler trois variables dans une formule non triviale.

Enoncé

J'ai 5 amis : Marie, Marc, Jean, Janine et Jean.
"Marie et Jean ne s'entendent pas"
"Marc et Marie ne viendront pas l'un sans l'autre
"Si j'invite Janine, il faut que j'invite Luc ou Jean, mais on ne peut pas les inviter tous les trois ensemble"
Après avoir déclaré 5 variables booléennes, chacune prenant la valeur vraie si la personne correspondante est invitée, écrire un programme qui permet de savoir si une invitation est possible.

Solution attendue

```
display                    'Marie
invité(e)  (O/N)'
read Rep
Marie :=(Rep='O' )
….
C1:=Not (Marie and Jean)
C2:=(Marc=Marie)
C3:=(Luc ≠ Jean) or not Janine
If C1 and C2 and C3 then
     display 'Possible'
 else
display 'Impossible'

          ……
```

Une solution "élève" typique

```
display                    'Marie
invité(e)  (O/N)'
read Rep
If Rep='O' then
   Marie :=true
else
    Marie :=false
…
If (Marie and Jean) then
     display 'Impossible'
else
     if  (Marc<>Marie) then
          display
     'Impossible'
     else…
```

Tableau 3 : un problème sur les booléens et deux amorces de solutions

Les solutions diffèrent d'abord quant à l'affectation d'une valeur logique aux variables booléennes après l'entrée d'une chaine de caractères. La solution attendue utilise l'affectation d'une condition à la variable, tandis que la solution "élève" passe par une alternative. Cette utilisation d'une alternative témoigne de ce que les conditions ne sont pas conçues par l'élève comme une formule dont le calcul a pour résultat une valeur logique. Par la suite, la solution attendue exprime les trois conditions et les affecte séparément à trois variables booléennes. Une alternative finale conditionnée par la conjonction des trois variables permet la sortie de la réponse. Pour l'expression des clauses, la solution "élève" typique privilégie à nouveau la structure alternative, en emboîtant plusieurs instances. Ceci permet une expression congruente à l'énoncé pour les deux premières conditions et évite le recours à la négation. Les solutions recueillies ne vont pas plus loin, car les élèves échouent à exprimer la troisième condition de cette manière. Notre interprétation est que les élèves conçoivent les expressions booléennes comme des "conditions" telles qu'elles s'expriment dans le langage usuel, plutôt que comme un calcul sur des valeurs logiques.

Nous ne détaillons pas davantage l'ingénierie. Lagrange (1991) montre une difficile

progression de ces élèves d'un bon niveau scientifique ayant été initiés au type booléen par leur professeur dans le cadre du programme de l'option. Ils restent attachés aux alternatives et très réticents à l'emploi de variables booléennes. Une progression souvent constatée est l'emploi d'un type connu. Par exemple les élèves déclarent une variable numérique à laquelle ils affectent 0 ou 1, ou une chaîne à laquelle ils affectent "OUI" ou "NON". Dans le premier cas, les conditions peuvent alors être calculées -par exemple (AxB=1) exprime la conjonction, ce qui constitue une étape vers la compréhension des booléens. Le second problème, repris par Lagrange (2002), vise à ce que les élèves construisent et utilisent le codage d'un graphe sous forme d'un tableau de booléen. L'ingénierie serait d'actualité par exemple dans le contexte des graphes et de l'algorithmique au lycée.

Quels savoirs sous-jacents ?

Nous avons commencé cette première partie en soulignant que l'expérience des années 1980 contredit l'idée d'un accès aisé des élèves à une expertise suffisante pour que les activités en programmation et algorithmique puissent contribuer aux apprentissages visés. Nous avons vérifié à l'aide d'exemples que ceci reste vrai dans le contexte actuel de l'algorithmique au lycée. Faisant le lien avec des exemples similaires dans le contexte de l'option informatique, nous avons analysé quelques aspects de cette question, que l'on peut résumer par la difficulté rencontrée par les débutants à prendre conscience des spécificités d'un langage informatique, c'est-à-dire plus précisément d'un langage destiné à être exécuté par un dispositif opérant en différé. Ceci se traduit notamment par une difficulté à interpréter les formes de ce langage comme exprimant un calcul sur des objets. Nous avons essayé de montrer que ces difficultés ne sont ni anecdotiques ni passagères. C'est pour nous l'indice qu'il existe des savoirs sous-jacents. Les exemples étudiés suggèrent un lien avec l'accès au symbolisme algébrique comme outil de modélisation et à la logique propositionnelle, mais sous une forme différente de la façon dont ces contenus sont abordés dans les mathématiques scolaires. Nous ne trouvons pas non plus ces savoirs ou concepts dans l'informatique telle qu'elle s'enseigne, l'expertise visée étant considérée comme un prérequis dans les cursus et ouvrages de référence. Dans la seconde partie, nous abordons la question des situations d'apprentissage, en mobilisant pour cela les outils de la didactique des mathématiques.

QUELLES SITUATIONS POUR LES PREMIERS APPRENTISSAGES ?

Dans la première partie, nous avons souligné les difficultés rencontrées par les débutants et leur similarité dans les deux contextes de l'option informatique dans les années 1980 et dans le contexte actuel de l'algorithmique au lycée. Nous avons montré que ces difficultés se manifestaient chez des élèves même après un premier enseignement tourné vers la programmation et esquissé quelques remédiations développées dans le premier contexte. Ce constat montre la nécessité, après avoir mieux cerné les savoirs et concepts sous-jacents à cette "alphabétisation", de réfléchir aux situations d'apprentissages. Nous le faisons à partir d'une synthèse opérée dans les 1980. Cette synthèse ne concerne pas l'option informatique qui n'a pas donné lieu à beaucoup de travaux critiques sur les pratiques en classe. Crahay (1987) s'est intéressé aux hypothèses émises, dans la lignée de Papert (1981), sur les apports des activités de programmation en LOGO à la "pensée procédurale", c'est-à-dire à la capacité de penser une série d'actions comme une procédure paramétrable et réutilisable dans un traitement plus complexe. LOGO est en effet un langage où l'on écrit des procédures qui peuvent s'appeler les unes les autres et de nombreux travaux de recherche ont recherché ses effets. Crahay fait une revue des évaluations, et conclut qu'elles ont toutes montré des résultats décevants. Il met en

cause particulièrement les pratiques des enseignants qui au départ laissent une grande liberté aux apprenants dans la construction de programme, mais ensuite se limitent à de interventions individuelles auprès des élèves et finissent par écrire le programme à leur place. Il conclut que les activités de programmation en Logo peuvent être l'occasion de "progrès cognitif incomparables", mais qu'il faudrait pour cela

> «1) une théorie de l'expertise qui décrit la performance terminale ou la structure de connaissance achevée que nous espérons faire acquérir par l'apprenant,
>
> 2) une théorie de l'acquisition qui décrit le processus de construction de la connaissance ou de la performance visée ou autrement dit les étapes par lesquelles les sujets transitent lorsqu'ils acquièrent cette compétence,
>
> 3) une théorie de l'intervention qui suggère quelles conduites d'enseignement sont susceptibles de stimuler le processus d'acquisition et qui fournit par le fait même une information utile pour savoir comment s'y prendre avec l'apprenant ». (Crahay, 1987, p. 53)

Nous souhaitons montrer dans cette partie comment la théorie des situations didactiques (Brousseau, 1988) a pu, dans certaines recherches, encore très modestes, répondre aux conditions posées par Crahay. Nous insistons en particulier sur la mise en place d'un milieu adéquat, à partir d'une analyse épistémologique ou psychologique des savoirs sous-jacents à l'alphabétisation et sur la mise en place de phases a-didactiques de confrontation des élèves à ce milieu. Il s'agit ainsi de rompre avec le constructivisme naïf dénoncé par Crahay, où partant de situations très ouvertes, l'élève est finalement conduit par l'enseignant pour l'écriture d'un programme.

D'une étude épistémologique du développement des machines et langages à une situation d'apprentissage de l'itération

La première recherche considérée ici est la thèse de Nguyen (2005) menée dans le contexte des programmes de mathématiques du lycée des années 2000. Nguyen s'intéresse à l'itération et montre que c'est une structure qui émerge dans l'évolution des calculatrices ou machines mathématiques de la calculatrice simple à la machine de Babbage. Cette dernière machine était conçue pour tabuler des fonctions de façon automatique, grâce à deux innovations : (1) un emplacement mémoire correspondant à une donnée invariante au cours d'un calcul répétitif peut prendre successivement des valeurs différentes, (2) le contrôle de l'exécution est fait non par l'opérateur, mais par la machine en fonction de l'état de la mémoire, ce qui rend notamment possible l'itération. La structure la plus récente est celle de l'ordinateur moderne où le processeur exécute un programme lui-même considéré comme une donnée, ce qui permet le contrôle de l'exécution par la machine de façon beaucoup plus souple. La situation d'apprentissage est basée sur cette étude. Il s'agit de confronter les élèves à la tabulation de fonctions dans des situations où il devient avantageux d'abord d'utiliser des mémoires, puis d'organiser l'activation des calculs et des mémoires dans une structure se répétant et finalement d'imaginer une instruction de répétition.

Les élèves ont des calculatrices simples, sans parenthèse, mais disposant de mémoires A, B, C pouvant stocker le résultat d'un calcul. La situation met aussi en scène un robot capable d'appuyer sur les touches de la calculatrice à partir d'une liste de touches fournie sur papier. La première tâche (T1, Tableau3) implique l'écriture fastidieuse d'une série de touches. Comme on le voit en partie droite, pour un pas de 0,2, il y a 26 séries de touches, chaque série commençant par l'entrée d'une valeur de la variable qui est stockée dans la mémoire A. Pour un pas de 0,03, il faudrait 166 séries de touches. Les élèves voient donc l'intérêt d'une suite invariante qui pourrait être répétée, ce qui est l'objet la tâche T2. Mais cela implique une

rupture, puisque dans une suite invariante chaque valeur de la variable ne peut plus être entrée individuellement. Dans la réponse donnée en partie droite du tableau 3, le calcul itératif des valeurs de la variable utilise une seconde mémoire B, l'élève ne voulant sans doute pas que la valeur dans la mémoire A soit perdue, bien que la suite invariante puisse utiliser une seule mémoire comme dans la réponse à la tâche T3. Dans cette dernière tâche les élèves doivent imaginer une structure dans laquelle insérer cette suite. Cette structure comprend une instruction spéciale de répétition quantifiée et une initialisation de la mémoire utilisée. On remarque dans la réponse donnée en partie droite que l'élève s'est trompé dans le nombre d'itérations : la valeur 1,98 est calculée, mais son image n'est pas affichée.

<table>
<tr><td colspan="2">Enoncé :"Soit f une fonction. Calculer les images par cette fonction de nombres espacés d'un pas donné dans un intervalle donné» f(x)=x²+1, intervalle [-3,2], pas 0,2 puis 0,03 Calculatrice avec mémoires STO</td></tr>
<tr><td>T1 Ecrire la suite des touches nécessaire pour la faire exécuter par un robot</td><td>- 3 Sto A A x A+1=
-2.8 Sto A A x A+1=
-2.6 Sto A A x A+1=
…
2 Sto A A x A+1=</td></tr>
<tr><td>T2 Ecrire un groupe de touches à faire répéter par le robot</td><td>Groupe de touches à répéter :
. A + 0,03 –> STO B
. B x B + 1 =
. B STO A</td></tr>
<tr><td>T3 Ecrire un message le plus court possible pour que le robot fasse tout le calcul de lui-même</td><td>-3 STO A
Répéter 166 fois A x A + 1 =
A + 0,03 STO A</td></tr>
</table>

Tableau3. Une situation d'apprentissage de l'itération : l'énoncé, les trois tâches (à gauche), et les réponses d'élève (à droite). D'après Nguyen et Bessot (2010).

La planification dans la construction d'un algorithme par des élèves

A la différence de l'exemple précédent qui partait d'une étude épistémologique, nous nous intéressons ici à une problématique issue de la psychologie de la programmation. Comment des débutants peuvent-ils construire un algorithme ou un programme tenant compte des spécificités d'un langage informatique ? Rogalski et Samurçay (1990) ont introduit cette problématique en distinguant les positions d'expert et de novice dans l'activité de "planification", c'est-à-dire de conception des étapes dans l'écriture d'un programme. Elles introduisent deux autres notions : les "schémas" qui sont les structures utilisées dans le traitement des informations pour atteindre des objectifs à petite échelle, et les "plans" qui sont des ensembles organisés de schémas. Les experts combinent généralement une approche descendante (top-down) où le plan est d'abord pensé, puis organisé en schémas élémentaires et une approche ascendante (bottom-up) qui part de schémas connus pour les organiser en plan. De façon générale si l'expert voit bien où il va, il construit un plan et ensuite s'intéresse à des sous-objectifs et aux structures correspondantes. Quand il voit moins bien comment commencer, il peut partir des schémas qu'il connait. Le

problème des débutants est qu'ils ne disposent pas d'un répertoire de schémas et que les plans auxquels ils pensent sont directement transposés d'une exécution manuelle ce qui rend difficile la spécification des sous-objectifs pour un dispositif informatique. Donc ils ne peuvent faire ni une approche descendante ni une approche ascendante.

Nous nous sommes intéressés à cette problématique pour l'encadrement d'un mémoire de master (Guy, 2013) et l'écriture d'un article (Lagrange & Guy, 2015) avec l'auteure du mémoire. Le contexte est celui de l'algorithmique au lycée et donc cette étude s'intéresse aussi aux interactions math-algorithmique. Un travail autour de l'algorithme de Kaprekar a été retenu de façon à mettre en jeu, à côté des savoirs mathématiques, des savoirs en programmation ou algorithmique, relatifs à la planification. Les savoirs mathématiques concernent la représentation des nombres, et plus généralement l'arithmétique.

1. Choisir un nombre entier de trois chiffres.
2. Former le nombre obtenu en arrangeant les chiffres du nombre choisi en 1. dans l'ordre croissant.
3. Former le nombre obtenu en arrangeant les chiffres du nombre choisi en 1. dans l'ordre décroissant.
4. Calculer la différence des nombres obtenus en 2. et 3.
5. Recommencer (à partir de l'instruction 2.) avec le résultat obtenu en 4, jusqu'à obtenir un nombre déjà obtenu.

Tableau 4 : L'algorithme de Kaprekar

Le traitement concerné est présenté dans le tableau 4. Il est discuté plus complétement par Lagrange et Guy (2015). Nous avons mis en place une situation didactique reposant sur le fait que dans ce traitement, le nombre courant est considéré sous deux représentations. Pour le tri (rangement dans l'ordre croissant ou décroissant) aux étapes 2 et 3, c'est un triplet de chiffres et pour la soustraction c'est un nombre au sens habituel du terme. A la main, la différence de représentation n'apparait pas. Un opérateur humain n'a pas conscience d'isoler des chiffres et de les ordonner aux étapes 2 et 3. Le traitement par une machine "arithmétique", c'est-à-dire disposant des 4 opérations dans les entiers et d'instructions de comparaison, impose de considérer le nombre courant sous ses deux représentations et donc de construire des étapes de conversion du nombre ordinaire en triplet et inversement. La situation s'adresse à des élèves de Terminale et la question qui leur est posée est celle de la terminaison de l'algorithme. Après qu'ils aient constaté sur quelques exemples que le traitement termine soit sur 0 soit sur 495, il est demandé aux élèves de concevoir un traitement dans leur environnement de programmation habituel de façon à examiner la généralité de ce constat[1]. La conception de la situation repose sur les hypothèses suivantes : (1) spontanément, les élèves vont s'engager vers un plan calqué sur l'exécution manuelle, car ils ne disposent pas des schémas de décomposition et de recomposition d'un nombre (2) par confrontation au dispositif "arithmétique" il peut se produire une prise de conscience de la nécessité d'aménager le plan du traitement, et plus généralement de ce qu'un plan convenant à un traitement manuel doit être adapté en fonction des capacités du dispositif destiné à l'exécuter.

Précisons le milieu. Il est constitué en premier lieu des données à traiter. Ici il s'agit d'objets mathématiques avec leurs propriétés arithmétiques qui exercent certaines rétroactions. Ce sont aussi des objets à codifier pour un traitement destiné à être exécuté par un dispositif exprimé dans un langage symbolique. Les rétroactions du langage résultent de contraintes d'expression et de représentation perçues de façon formelle, (cette écriture n'est pas "conforme") ou en référence à un dispositif (l'ordinateur ne va pas comprendre) et souvent les deux à la fois.

Nous résumons ici les choix pour la mise en œuvre : les fonctionnalités du langage sont

[1] Un prolongement, mis en place dans le cadre d'un travail de thèse concerne la preuve de de ce résultat (Laval 2015).

limitées aux fonctions arithmétiques et au traitement de listes, la structure itérative est du type `tant que` avec une condition de continuation sur deux valeurs, ce qui est plus simple que le repérage d'un invariant ou d'un cycle et permet de laisser la conception de cette condition à la charge des élèves (voir la première partie.) Un autre choix essentiel est de mettre l'accent sur la représentation des données (choix des variables) pour engager une dynamique de planification productive : pensant à un plan calqué sur l'exécution manuelle, les élèves vont d'abord se contenter d'une variable pour le nombre courant et éventuellement de variables pour les nombres obtenus par rangement croissant et décroissant des chiffres; il est possible que certains élèves perçoivent que, pour un traitement par une machine, il est nécessaire de prévoir les variables sur lesquelles opérer le rangement des chiffres du nombre courant ; si ce n'est pas le cas, l'enseignant peut enclencher cette réflexion en demandant aux élèves si leur choix permettrait le traitement par exemple d'un nombre à 10 chiffres. La prise de conscience de la nécessité de variables pour les chiffres devra ensuite engendrer le besoin de sous-algorithmes de décomposition et de recomposition d'un nombre.

Les 4 phases en résultent de ces choix. Dans une première phase, il est demandé aux élèves quelles variables sont à déclarer, et on s'attend, comme nous venons de le dire, à ce que se produise une prise de conscience de la nécessité d'introduire une ou des variables pour les chiffres du nombre courant, soit sous forme d'une liste à trois éléments numériques, soit plus simplement avec trois variables numériques. Ensuite on peut penser que cette prise de conscience va être partagée dans la classe et qu'un plan incluant des sous-algorithmes de conversion va être identifié. Dans une seconde phase, les élèves conçoivent les trois sous-algorithmes de la figure 1. Dans un esprit de modularité, les sous-algorithmes sont confiés à des groupes différents. La troisième phase consiste à rédiger l'algorithme complet en organisant les sous-algorithmes dans une itération. Il ne s'agit d'une simple concaténation. En effet les sous-algorithmes ont des déclarations de variables spécifiques qui doivent être harmonisées (figure 1). De même pour le nombre entré dans le premier sous-algorithme doit être harmonisé avec le nombre rendu par le 3ème. Les élèves ont aussi à construire l'itération avec une condition d'arrêt adéquate (épisode discuté en partie I).

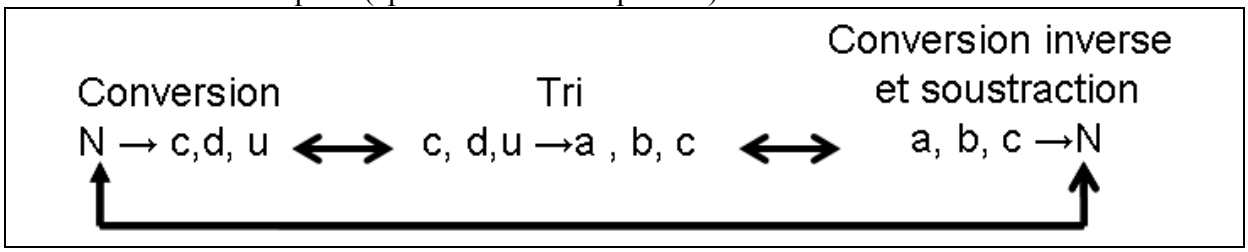

Figure 1 : Les trois sous-algorithmes du corps de boucle et l'harmonisation des variables

Voici quelques éléments d'analyse à posteriori dans chacune des phases. Dans la première phase, nous assistons bien à une prise de conscience de la nécessité de deux représentations sous l'influence du dispositif. Pour l'illustrer, voici deux prises de parole successives par deux élèves, l'un fait référence au traitement manuel "on va prendre" et présente un choix naïf des variables, dans un schéma itératif. Le second élève fait lui référence au dispositif ("il") et à ses capacités et identifie la nécessité de variables pour les chiffres.

> **E1. On va prendre** A en nombre initial, **on va prendre** B arrangé en ordre décroissant, C en ordre croissant, **on va faire** la différence de, euh, C moins B. O**n va le remettre** dans A, et **on va refaire** avec B et C.
>
> **E2.** Ben si A c'est une variable où c'est un nombre, pour qu'**il** puisse le ranger dans l'ordre croissant et décroissant, faut d'abord séparer les chiffres des dizaines, des unités et des centaines. Donc il faut trois variables….

Comme prévu aussi les élèves dans une phase collective identifient les étapes et par groupe sur

ordinateur conçoivent les sous-algorithmes. Les solutions sont variées, et là aussi on constate à nouveau que le dispositif joue un grand rôle dans la réflexion des élèves. Dans la dernière phase, on rencontre la difficulté attendue à assembler les sous-algorithmes avec des variables cohérentes entre les sous-algorithmes, mais aussi en entrée et en sortie du corps de boucle (figure 1). Certains élèves ont construit des solutions pour plusieurs sous-algorithmes à la seconde phase et rencontrent moins de difficulté à les assembler, ce qui indique que la modularité, c'est-à-dire la capacité à considérer un traitement par une spécification abstraite plutôt que par sa réalisation, ne va pas soi.

L'apport de la théorie des situations didactiques

L'ingénierie proposée par Nguyen a certes ses limites puisque l'itération se limite à la répétition et reste liée au contexte de la tabulation de fonctions. Néanmoins, elle met bien en valeur l'idée de variable itérative, comme mémoire (physique) d'une machine mais aussi comme signifiant dans l'écriture d'un traitement. Les élèves sont face à une tâche qui a joué un rôle dans le développement de l'informatique et ils doivent construire des éléments clés d'une machine informatique et du langage de commande permettant d'y répondre de façon efficace. Nous avons observé dans l'option informatique que l'itération est présentée aux élèves par monstration. Le professeur montre un programme et explique comment il fonctionne, avant de proposer des tâches similaires. Il n'est pas étonnant que, sans construction, de nombreux élèves soient en difficulté. Un point crucial est celui des relations entre les opérations d'initialisation et celles constituant le corps de boucle. Selon Samurçay (1985, p. 148) les relations entre ces deux éléments de l'itération sont particulièrement complexes et même après une première initiation, cette complexité entraîne des confusions persistantes chez les débutants. Un des points forts de l'ingénierie, lié à la construction par les élèves d'une structure en réponse à un problème, est qu'ils distinguent bien les opérations d'initialisation de celles constituant le corps de boucle (voir Nguyen, 2005, notamment p. 225).

Tout comme l'ingénierie de Nguyen, la situation de l'algorithme de Kapreka montre les bénéfices d'une approche "théorie des situations didactiques". Les élèves ont progressé seuls sur les points critiques : prise de conscience de le nécessité de sous-algorithmes de conversion, conception et assemblage des sous-algorithmes, et plus généralement la nécessité de penser un plan, non en fonction d'un traitement manuel, mais pour l'expression dans un langage informatique. Comme nous l'avons indiqué plus haut (note 1), une quatrième phase visant à l'étude de la terminaison de l'algorithme a été mise en place dans le cadre d'un travail de thèse. L'analyse de cette phase par Laval (2015) montre que ce travail en a-didacticité sur l'algorithme donne aux élèves une conception des objets en jeu adéquate pour cette phase centrée sur la preuve. Une approche "théorie des situations didactiques" n'apporte pas seulement un milieu offrant une interaction efficace. La conception même du milieu conduit à penser les savoirs en jeu en fonction de cette interaction. Pour l'itération, il faut répondre à des questions telles que : Quels problèmes font ressentir la nécessité d'une structure itérative ? Quelles étapes sont à franchir ? Comment le langage évolue-t-il avec ces étapes ? Les savoirs liés à la notion de variable informatique évoluent ainsi parallèlement à l'évolution des capacités de la machine. Le milieu pour la planification comprend quant à lui un objet sous deux représentations. Les savoirs sous-jacents sont la capacité à concevoir ces deux représentations, à déterminer des variables adéquates pour cela et à assurer une gestion cohérente de ces variables dans l'assemblage de sous-algorithmes dans un corps de boucle.

LES ACQUIS DE LA RECHERCHE EN PSYCHOLOGIE DE LA PROGRAMMATION ET EN DIDACTIQUE DE L'INFORMATIQUE

Comme nous l'avons dit en introduction, l'enseignement de l'algorithmique et de la programmation est revenu dans les programmes sans référence à l'expérience acquise antérieurement, en particulier à propos de l'option informatique des lycées (en gros, une décennie entre 1981 et 1992), et sans exploitation du questionnement déjà ouvert alors sur les apports potentiels de l'enseignement de l'informatique pour l'apprentissage mathématique (essentiellement au niveau du lycée). Les collègues moins anciens que nous ignorent sans doute les recherches qui se sont parallèlement développées en didactique de l'informatique et en psychologie de la programmation (en particulier à propos des débuts dans le domaine). Ce sont ces acquis sur lesquels cette partie veut attirer l'attention, en faisant un point rapide sur les mouvements qui ont eu lieu dans ce domaine, en soulignant un certain nombre d'acquis stables sur l'activité de programmation et sur les concepts et obstacles épistémologiques et en relevant l'existence de deux questions récurrentes : celle des précurseurs potentiels de la programmation et celle des impacts mutuels entre informatique et mathématique.

L'évolution du domaine

L'évolution du domaine a été présentée en détail lors du Congrès de 2013 de didactique de l'informatique (Rogalski, 2015). Nous en donnons ici un bref résumé. Dans les années 1970, des théoriciens de l'informatique ont recherché des concepts puissants de programmation. Ensuite est venue une "seconde génération", celle d'études expérimentales d'informaticiens largement tournées vers les propriétés souhaitables des langages informatiques. La décennie des années 1980 a vu le déploiement des études en psychologie de la programmation (particulièrement – mais pas seulement - des débutants) et l'émergence d'une didactique de l'informatique (sous ce nom-là dans la communauté francophone européenne) pour une "alphabétisation" scolaire dans ce champ. On observe ensuite, dans une "troisième génération" un "rétrécissement" des recherches sur l'informatique scolaire avec en contrepoint une ouverture des recherches de psychologie de la programmation vers les initiations du niveau universitaire ou professionnel. En conséquence l'intérêt s'est porté vers les langages "orientés objets" (LOO) alors que les langages "procéduraux" dominaient dans les recherches sur le secondaire. Il faut cependant relever une continuité de recherches en psychologie de la programmation via le groupe "The Psychology of Programming Interest Group" (PPIG) constitué à Cambridge (UK) en 1987 pour réunir des acteurs de diverses communautés pour explorer des intérêts partagés dans les aspects psychologiques de la programmation et dans les aspects "computationnels" de la psychologie.

La quatrième génération (après les années 2000) s'est orientée vers une didactique pour l'informatique universitaire et professionnelle – au sens où les questions traitées sont d'une part celles des premiers enseignements universitaires, et d'autre part, celles des acquisitions de concepts avancés, en relation en particulier avec le développement de la programmation "parallèle". Deux ordres de raisons semblent à l'origine de ces nouvelles recherches, essentiellement conduites par les informaticiens enseignant à des niveaux post-secondaire : le manque menaçant d'informaticiens dans les pays occidentaux et le besoin d'avoir des professionnels du niveau exigé pour la sécurité et la fiabilité des programmes à concevoir dans nombre de domaines sensibles (dont ceux touchant les bases de données), alors que les étudiants se dirigent moins vers les études scientifiques en général et l'informatique en particulier. Une triple contrainte semble apparaître : il faut attirer des étudiants en initiation à l'informatique, les garder pour la poursuite d'études dans ce champ – ce qui suppose qu'ils réussissent "suffisamment", et les préparer à être des informaticiens répondant aux attentes de la qualité de

la programmation – ce qui suppose qu'ils atteignent un niveau "suffisant" dans leurs acquisitions. Le retour de l'informatique dans la scolarité du secondaire apparaît un moyen de préparer un "vivier" d'étudiants qui permette de répondre à ces contraintes, comme l'exprime l'introduction d'un papier récent :

> While needs of high school students in the US are being prioritized through courses such as Exploring Computer Science (ECS) (…), there is a growing belief that experiences with computing must start at an earlier age. (Grover, Pea & Cooper, 2016).

Le questionnement proprement didactique (élaboration et conduite de situations appropriées aux acquisitions visées, pour dire vite) est toutefois resté limité par rapport à celui orienté vers ces acquisitions, y compris en France. Un article (da Rosa, 2015), d'orientation piagétienne et se référant à la Théorie des Situations Didactiques, regrette d'ailleurs lors du dernier congrès de PPIG que cette dimension "didactique" reste marginale dans les recherches au niveau international. Pourtant, comme l'illustrent les exemples développés dans la première partie de cet article, non seulement on observe une persistance des questions pertinentes soulevée par la seconde génération de recherches, mais on relève aussi la validité des résultats de la recherche des années 1980, dont nous allons rappeler les grandes lignes.

Les acteurs

Il convient de citer quelques acteurs qui ont marqué ce domaine dans la seconde génération en France et dont les travaux ont été repris comme références dans la dernière … lorsqu'ils étaient publiés en anglais. Pour la didactique, je citerai particulièrement André Rouchier et Renan Samurçay. Le texte de Rouchier : "Objets de savoir de nature informatique dans l'enseignement secondaire" situe l'informatique dans un champ d'interrogation didactique, puis analyse les objets informatiques en jeu dans l'écriture de programmes élémentaires, objets d'un domaine de savoir qui peut constituer un "habitat" nouveau pour des savoirs mathématiques (Rouchier, 1990). On peut le reprendre comme un manifeste didactique toujours d'actualité. Il s'appuie sur un ensemble de recherches, dont la plus emblématique en didactique a été conduite avec Renan Samurçay avec une approche de la structure récursive comme réponse à un certain type de problèmes (Samurçay & Rouchier, 1990). La disparition d'un enseignement de la programmation au niveau scolaire n'a pas permis à ce travail conceptuel de didactique d'avoir l'impact qu'il aurait mérité.

En revanche, des acquis sur la psychologie de l'activité de programmation portant sur les concepts informatiques et les activités cognitives en jeu dans la programmation et concernant tout programmeur débutant (et pas seulement les élèves en contexte scolaire) sont restés "vivants". Leur caractère conceptuel - non centré sur la dimension "langage" de la programmation - a certainement joué, et bien sûr le caractère international de leur langue de publication. Il en est ainsi de la recherche de Renan Samurçay sur le concept de variable informatique sur lequel on revient plus loin (Samurçay, 1985/1989), dont la version anglophone est une référence toujours citée (comme l'ouvrage édité par Soloway et Spohrer dans lequel elle est publiée). Un autre acteur de l'organisation de recherches sur la programmation, Jean-Michel Hoc, a également contribué à la production d'un ouvrage de synthèse "Psychology of programming" mis en ligne pour les étudiants en informatique de Cambridge comme référence sur la dimension humaine de la programmation (Hoc, Green, Samurçay & Gilmore, 1990).

Les acquis sur l'acquisition des concepts et l'activité de programmation

Après les acteurs, faisons un point sur les acquis des recherches de la décennie 1980. Ces acquis sont de nature épistémologique et cognitive. Ils sont relatifs aux spécificités de la discipline : l'informatique est à la fois science et pratique (on parle de génie informatique, comme on parle

par exemple de génie civil pour l'ingénierie du bâtiment), et à ce titre elle pose un double problème de transposition dans l'enseignement ; l'activité cognitive de programmation présente des spécificités ; des concepts-clés ont des précurseurs qui peuvent jouer un rôle producteur et un rôle réducteur ou sont sans précurseurs ; des connaissances opérationnelles particulières sont en jeu pour la programmation, en particulier des méthodes.

Ces spécificités ont des implications didactiques. Tout d'abord, le double problème de transposition de la pratique de la programmation et du savoir informatique comme savoir scientifique se manifeste dans la conception de situations didactiques. On peut le caractériser par la question suivante : comment articuler l'élaboration de situations didactiques centrées sur l'acquisition de concepts déterminés (variables, boucles, conditionnelles) - dans des problèmes bien circonscrits comme dans les exemples des première et deuxième parties, donc "jouets" - avec une activité de programmation partant - comme la programmation professionnelle - de problèmes consistants, ouverts et suffisamment "attrayants" pour les élèves, dans des situations de "projet" ? Concernant ces situations de "projet", le constat général des études récentes comme de celles conduites autrefois - en particulier avec LOGO - est le fait que l'action didactique de l'enseignant est cruciale pour que l'engagement des élèves ou étudiants se traduise par l'acquisition de concepts de programmation. La nature de cette action enseignante reste à notre point de vue une question didactique ouverte.

En second lieu, l'analyse des spécificités de l'activité de programmation informatique a conduit à souligner plusieurs points :

- en programmation il s'agit de passer de "faire" à "faire faire" (Samurçay & Rouchier, 1985) : l'exécution est déléguée à une "machine", avec perte de contrôle de celui qui a conçu le programme ; cela implique de se constituer une représentation opérationnelle de la machine ou plus précisément du "dispositif informatique" (Rogalski, 1988 ; Rogalski & Hé, 1989) ou de la *notional machine* (Du Boulay, 1986).
- il faut effectuer un changement de niveau entre réalisation d'une occurrence particulière d'une procédure et élaboration d'une procédure générique : cela suppose de passer d'une connaissance "en acte" à une analyse des objets et des actions en jeu ; il s'agit aussi de se questionner sur les entrées qui peuvent être pertinentes, sur la validité de la procédure selon le domaine sur lequel elle va "tourner" ;
- l'analyse des objets et des actions possibles dans le dispositif informatique appelle des acquisitions sur le "système de représentation et de traitement" des données informatiques (SRTI), dont Lagrange (1991) a montré combien c'était un processus difficile, en particulier pour les booléens, en jeu dans les conditionnelles comme dans les boucles et la récursivité;
- il y a besoin d'un contrôle logique des conditionnelles et des itérations - et non plus du contrôle sémantique, qui est d'ailleurs souvent présupposé et non explicité dans les exécutions "à la main". Ainsi, dans une conditionnelle, le dispositif informatique est censé "comprendre" que lorsque le traitement d'une condition A (si A alors faire) est terminée la suite du programme concerne la condition non A (sinon ...), et que lorsque les deux cas ont été traités on passe à la suite de la procédure (en sortant de la conditionnelle) (Rogalski & Hé, 1989) ;
- dans la conception d'un programme interactif destiné à un utilisateur (peut-être le concepteur lui-même, mais plus tard), la représentation du triplet < "je conçois", " un dispositif informatique exécute", "un humain utilise"> appelle une articulation de la procédure "noyau" du programme avec des opérations d'entrée de données (une lecture pour le dispositif...) et de sortie que va lire l'utilisateur (un affichage pour le dispositif), opérations dont la formulation ne va pas de soi lors de l'alphabétisation .

En troisième lieu, des difficultés, voire des obstacles, se présentent régulièrement dans l'acquisition des deux grands concepts clés lors de l'enseignement des bases de la programmation (ce qu'on a appelé l'alphabétisation) : (1) la variable informatique, qui est ...

une fonction (de l'exécution), avec des rôles différenciables dans le programme, nous y revenons (2) la notion d'invariant de boucle (au cœur des écritures récursives), une clé pour la validité des programmes : elle est difficile à dégager - même en acte - pour l'écriture de boucles, avec la nécessité de mettre en relation la valeur des données dans l'entrée dans la boucle, l'exécution du corps de boucle et la valeur des données à la sortie de la boucle. Le contrôle de la fin de l'itération ou de la procédure récursive est une difficulté particulière pour les débutants (voire au-delà ...) Dans l'étude récente citée plus haut d'un enseignement des bases de la programmation au niveau collège ("middle school" US) les auteurs soulignent :

> Serial execution was the easiest to learn, as expected. Between conditionals and loops, learners found loops harder to tackle. Most of the assessment questions concerning loops required manipulation of variables as well, which seemed to be the hardest topic for students to grasp. [...] Both these aspects have been known to be particularly difficult for novice programmers (…) Despite our conscious efforts, students struggled with these topics. (Grover, Pea & Cooper, 2016).

Ces constantes ont été relevées dans des introductions faites le plus souvent dans une approche algorithmique, utilisant des rédactions en "pseudo-code". D'autres difficultés/obstacles spécifiques ont été relevés pour la programmation en langage orienté objet (LOO), dont les relations entre classes et les processus d'héritage. Mais l'écriture des "méthodes" des objets fait rencontrer aux élèves les mêmes notions clés de variables, conditionnelles et boucles. Il faut souligner l'existence de multiples interactions, d'une part entre les concepts informatiques eux-mêmes, et d'autre part entre les concepts et les activités cognitives. Ainsi, une des difficultés de la conception de programmes utilisant la récursivité est la nécessité de faire des "raisonnements sous hypothèse" , dont le schéma est alors le suivant lors de l'écriture d'un module récursif ou itératif (pour simplifier, récursivité sur n entier) : "je suppose que j'ai traité mon problème jusqu'à n, le problème actuel est de trouver comment passer de n à n+1 [c'est justement l'invariant de boucle], puis de chercher quelle valeur de n me permet d'initialiser le processus". Le mode spontané des élèves (pas seulement du secondaire) est de partir de n=1, de chercher à passer à 2, et de coder une forme de "etc.". Nous renvoyons à Samurçay et Rouchier (1990) pour une approche didactique de la récursivité lors de l'alphabétisation informatique. Les structures itératives comme récursives illustrent bien le fait que la variable informatique est une fonction de l'exécution. Ce n'est pas sa seule complexité conceptuelle.

La variable est un concept difficile

Nous avons vu que la variable informatique est une fonction, au sens où sa valeur dépend du moment de l'exécution du programme (évidemment, cette fonction peut être constante, si la variable ne change pas de valeur au cours de l'exécution). Ceci la différencie de la variable mathématique qui peut cependant servir de précurseur : elle a un rôle producteur, via la manipulation formelle de représentations algébriques, mais aussi un rôle réducteur, par son caractère de permanence au cours d'un traitement (dans la résolution de l'équation ax+b=c, x est toujours "le même" et cette invariance est encore plus importante pour résoudre un système d'équations). L'appropriation de la notion de fonction (en mathématique) peut limiter ce rôle réducteur, ce qui pourrait d'ailleurs contribuer à expliquer l'impact du niveau mathématique des élèves sur leurs acquisitions en informatique.[2]

[2] L'importance de la notion de variable en programmation informatique a conduit des chercheurs à élaborer un test sur l'interprétation de relations impliquant l'affectation à des étudiants allant entrer dans leur premier cours d'informatique et ignorant de la programmation. Une méta-analyse des résultats (Dehnadi, Bornat & Adams, 2009) a montré qu'une bonne cohérence dans l'interprétation de l'opération d'affectation (avant toute introduction à un langage informatique) présageait de meilleur succès dans le début de l'apprentissage de la programmation (mais que leur épreuve ne pouvait pas servir valablement de test de sélection). On pourrait faire l'hypothèse ce qui est en

D'autre part, le champ conceptuel de la variable informatique fait intervenir les notions de type de variable, la représentation dans une donnée structurée, et l'existence d'un ensemble d'opérations sur la variable. Il comporte aussi des rôles fonctionnels de la variable dans le programme. Enfin, dans la mesure où un programme suppose une modélisation du problème en termes informatiques, les relations des variables utilisées avec les composants du problème sont plus ou moins délicates (cf. les exemples traités dans les deux premières parties). Une analyse de ce champ conceptuel a été développée pour l'alphabétisation par Samurçay (1985/1989) et pour la programmation dans l'enseignement supérieur Sajaniemi (2008) et Sorva (2008), qui font référence à Samurçay (1989). Samurçay a distingué 4 rôles de la variable, lors de l'alphabétisation (entre parenthèses les termes retenus par Sajaniemi) : (1) donnée (fixed value), (2) compteur (stepper) (3) accumulateur (gatherer) (4) intermédiaire pour la programmation (temporary). Pour la programmation plus avancée, Sajaniemi a enrichi cet herbier de rôles, en identifiant 6 autres rôles, qui avec les précédents couvrent plus de 99% des rôles rencontrés dans l'enseignement de la programmation, qu'elle soit procédurale ou orientée-objet. Certains de ces rôles s'actualisent dans le traitement de données comme des listes, des arbres, des tableaux. Les propriétés des variables et de leur traitement qui présentent des accès plus ou moins délicats pour les élèves sont leur statut dans la modélisation : une variable qui a un rôle d'intermédiaire présentant du sens dans le problème (externalité de la variable) est plus accessible cognitivement que celle nécessitée par l'existence du dispositif informatique (internalité de la variable) ; un compteur qui reflète les étapes d'un traitement "à la main" est plus aisé à représenter et utiliser, de même qu'un accumulateur. L'impact des rôles interfère avec les types de données et les opérations en jeu (en particulier l'initialisation et le test - par sa nature, la structure de donnée retenue, sa place dans une boucle ...). Au-delà de l'alphabétisation, Sorva (2008) a souligné par ailleurs que :

> The behavior of a variable, that is the logic that dictates how the variable is used, is often defined at multiple distinct locations in program code [...], may be described by inconsecutive lines of code within a function or a method, located in a number of functions or even located in several program modules (p. 64).

Il a également montré que si on explicite aux étudiants les rôles des variables dans les programmes, ils font davantage de progrès en programmation.

Difficultés relatives des activités cognitives portant sur le champ conceptuel de la variable informatique

Parmi les activités cognitives spécifiques à la programmation informatique, qui posent problème au débutant, se pose la constitution d'une représentation des différentes opérations que l'on peut effectuer sur les composants du programme, en particulier les variables auxquelles on s'intéresse ici. La déclaration de variable (pas toujours imposée par le langage) peut contribuer positivement à la construction d'une représentation du système de représentation du langage (à un niveau "logique") elle peut contraindre aussi à rencontrer des types de variable non familiers à l'élève. Nous avons montré en première partie le saut cognitif que constitue l'utilisation de booléens - et de leur combinatoire – pour exprimer des conditions.

Les registres de représentation utilisés en mathématiques peuvent ne pas suffire à traiter informatiquement un problème, dès lors que celui-ci fait intervenir d'autres registres dont le sujet n'est pas conscient parce qu'il les a "naturalisés". Il en est ainsi dans la lecture d'un nombre entier à la fois comme suite de ses chiffres et comme nombre objet potentiel des "quatre opérations". L'exemple de l'algorithme de Kaprekar développé en seconde partie illustre cette

jeu n'est pas tant le concept de variable (que les étudiants construiront plus ou moins bien au cours de leur alphabétisation informatique) que la manipulation d'un registre symbolique arbitraire.

problématique - en même temps qu'il constitue une situation didactique appropriée pour donner du sens à la notion de type de variable (qu'on utilise ou non ce terme). Rôle et déclaration sont des éléments indépendants dans le champ conceptuel de la variable ("declaring a variable, if it is explicitly done in the language at all, is a matter separate from the variable's behavior" - Sorva, 2008). Le contrôle des valeurs dans le programme suppose de s'affranchir des présuppositions des traitements à la main. On peut voir dans les exemples d'écriture du programme de résolution de l'équation du premier degré ax+b=c, l'absence de prise en compte du cas a=0 : le contrat didactique courant appelle à éventuellement préciser $a \neq 0$ dans l'expression du calcul x=(c-b)/a, sans que le statut du cas a=0 soit pour autant repéré: la présupposition sous-jacente est que si on parle de résoudre l'équation du premier degré c'est que la variable x apparaît effectivement dans l'expression, ce qui n'est pas le cas si a=0.

De manière générale, les opérations sur les variables - initialisation, mise à jour, test - peuvent constituer en elles-mêmes des ruptures avec la pratique habituelle des élèves, même avec l'écriture d'un algorithme "papier-crayon", encore plus dans l'exécution d'un algorithme, lors de laquelle l'élève sait bien quelle est la valeur de la variable qu'il traite... L'organisation séquentielle des opérations sur les variables peut aussi être en rupture avec les traitements "à la main" - ce qu'on a observé dans la difficulté plus grande à utiliser des boucles conditionnelles plutôt que des itérations en nombre déterminé (pas de test explicite), la difficulté plus grande à maîtriser les boucles de forme "tant que" (où on teste une variable avant tout traitement) par rapport aux boucles de forme "jusqu'à" (le test sur la variable est un test d'arrêt en position finale dans la boucle). Le contrôle des valeurs prises par une variable au cours de l'exécution d'un programme est une activité cognitive qui n'est pas non plus dans la continuité des opérations "à la main" ; ici le traitement logique des variables dans le programme interagit avec les représentations de l'exécution du programme. En fait, une propriété fondamentale sous-jacente à nombre de ces difficultés cognitives de représentation et de traitement de la variable informatique est que celle-ci est ... une fonction. On pourrait la rapprocher de ce qui est rencontré plus tard par les élèves (dans le secondaire, en France), à savoir la notion de suite numérique - dont on peut se demander quand elle a pour eux le statut de fonction.

CONCLUSION

Nous avons insisté dans cet article sur les difficultés conceptuelles des élèves, souvent ignorées ou au mieux mal connues des acteurs du terrain ou institutionnels. Nous avons montré qu'elles se rencontrent très généralement chez les débutants dans différents contextes temporels et curriculaires. Considérer ces difficultés ne veut en aucun cas dire qu'un enseignement d'algorithmique ou de la programmation à des débutants serait vain. Nous sommes impliqués dans le domaine depuis plus de trente ans et l'évolution sociale confirme ce que nous pressentions à nos débuts : il est important que les élèves puis les adultes soient conscients de l'activité humaine à l'origine de toute application informatique, et de la rationalité qui préside à cette activité. Le repérage des difficultés nous conduit à penser qu'il est utile que l'initiation commence tôt de façon que les processus de maturation conceptuelle des élèves puissent avoir lieu sur le temps long des études.

La didactique a un rôle très important à jouer. Nous avons montré les nombreux acquis côté "psychologie de la programmation", mais, dans le contexte d'un enseignement scolaire de l'informatique qui serait lié aux mathématiques, le champ des questions ouvertes reste très vaste. Quel est précisément l'étendue des difficultés ? Dans quelle mesure affectent-elles les apprentissages visés ? Quels précurseurs peuvent jouent un rôle producteur en permettant aux élèves qui en disposent de surmonter rapidement ces difficultés, ou un rôle réducteur inverse ? Nous avons souligné aussi la quasi absence de travail proprement didactique : concevoir,

conduire et analyser des situations didactiques candidates à des acquisitions des élèves. Ceci renvoie à une époque où les travaux initiaux en didactique des mathématiques et en didactique des sciences étaient centrés soit sur des approches purement épistémologiques, soit sur les "erreurs" ou "misconceptions".

Les convergences entre les mathématiques et l'informatique sont souvent soulignées au plan de l'épistémologie. Mais qu'en est-il au plan des apprentissages ? Des études montrent régulièrement que des élèves qui réussissent en maths ont une meilleure entrée dans la programmation - mais il s'agit d'évaluations assez globales, qui ne font pas directement apparaître ce qui pourrait être sous-jacent. Y-a-t-il réellement dans les connaissances mathématiques des précurseurs producteurs en programmation ? Ou s'agit-il d'une corrélation dont des précurseurs communs, comme par exemple de bonnes représentations de l'espace, ou une bonne manipulation des opérations de raisonnement, rendraient compte ? Algorithmes ou programmes fournissent un registre de représentation supplémentaire pour de nombreuses entités mathématiques –figures, formules, fonctions… Quelles activités de programmation permettent effectivement à ces registres (Duval, 1995), ainsi que les traitements et conversions qui leur sont attachés, de devenir disponibles chez les élèves, et conduiraient ainsi à des effets en retour de l'apprentissage de la programmation sur les acquis en mathématiques ?

La didactique des mathématiques a aujourd'hui des outils multiples, théoriques et méthodologiques, pour aborder les questions que nous venons de soulever. Les exemples de la seconde partie montrent l'intérêt d'une approche dans le cadre de la Théorie des Situations Didactiques, encore peu utilisée dans le secondaire il y a 25 ans. Cette approche conduit à concevoir un milieu offrant à l'élève une interaction efficace et oblige à penser les savoirs en jeu non seulement au plan épistémologique ou cognitif mais aussi au plan proprement didactique. Les travaux sur la dimension sémiotique dans l'activité mathématique, parmi lesquelles la notion de registre de représentation que nous venons de mentionner tient une place importante, pourraient apporter leur contribution dans un domaine où les écritures symboliques tiennent une grande place. Au-delà des situations "simplifiées" visant spécifiquement à faire affronter des concepts clés de la programmation, les travaux sur la modélisation pourraient contribuer à l'étude de situations de "projets" visant à transposer la pratique informatique. Un champ de recherche très large est ouvert. Nous espérons que cet article sera utile pour cela.

REFERENCES BIBLIOGRAPHIQUES

BARON, G.-L. (1989) *L'informatique discipline scolaire*. Paris : PUF.

BRIANT, N. (2013) *Étude didactique de la reprise de l'algèbre par l'introduction de l'algorithmique au niveau de la classe de seconde du lycée français*. Thèse Université Montpellier II - Sciences et Techniques du Languedoc.

BROUSSEAU, G. (1988) *Théorie des situations didactiques*. Grenoble : La Pensée Sauvage.

CRAHAY, M. (1987) Logo, un environnement propice à la pensée procédurale ? *Revue française de pédagogie*, 80(1), 37-56.

DA ROSA, S. (2015) The construction of knowledge of basic algorithms and data structures by novice learners. In M. Coles & G. Ollis (Eds.), *Proceedings of the Psychology of Programming Interest Group Annual Conference 2015* (pp. 37-47). http://ppig.org/sites/default/files/2015-PPIG-26th-proceedings.pdf (consulté 17/2/2016).

DEHNADI, S., BORNAT, R., & ADAMS, R. (2009) Meta-analysis of the effect of consistency on success in early learning of programming. Proceedings of the 21th PPIG Conference.

DU BOULAY, B. (1986) Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1), 57-73.

DUVAL, R. (1995) *Sémiosis et pensée humaine*. Bern: Peter Lang.

ENGEL, A. (1979) *Mathématique élémentaire d'un point de vue algorithmique* : (adapté par Daniel Reisz), Paris : CEDIC.

GROVER, S., PEA, R.D., & COOPER, S. (2016) Factors influencing computer science learning in middle school. *SIGCSE '16*, March 02-05, Memphis, TN.

GUY, M.-N. (2013) *Utilisation du cadre théorique de la planification pour la conception d'algorithmes complexes par des élèves de lycée*. Mémoire de master de Didactique des mathématiques. Université Paris Diderot.

HOC, J. M., GREEN, T. R. G., SAMURCAY, R., J. GILMORE D. J. (Eds) (1990) *Psychology of programming*. London: Academic Press.

LAGRANGE, J.-B. (2002) Les outils informatiques entre "sciences mathématiques" et enseignement. Une difficile transposition ? In Guin D. & Trouche L. (eds) (2002). *Calculatrices symboliques, faire d'un outil un instrument du travail* mathématique (pp. 89-116). Grenoble : La Pensée Sauvage.

LAGRANGE, J.-B., & GUY, M.-N. (2015) Planification et connaissances mathématiques dans une situation d'apprentissage au lycée : l'algorithme de Kaprekar. *Petit x*. 97, 45-70.

LAGRANGE, J-B. (1991) *Des situations connues aux traitements sur des données codifiées : représentations mentales et processus d'acquisition dans les premiers apprentissages en informatique*. Thèse de Doctorat. Université Paris 7.

LAVAL, D. (2015) L'algorithmique comme objet d'apprentissage de la démarche de preuve en théorie élémentaire des nombres : l'algorithme de Kaprekar. *Actes du Quatrième symposium international Espace de Travail Mathématique*, pp. 103-116. Madrid : I.M.I.

NGUYEN, C. T. (2005) *Étude didactique de l'introduction d'éléments d'algorithmique et de programmation dans l'enseignement mathématique secondaire à l'aide de la calculatrice*. Thèse de l'université Joseph Fourier, Grenoble.

NGUYEN, C. T., BESSOT, A (2010) Introduire des éléments d'algorithmique et de programmation dans l'enseignement secondaire ? Une ingénierie didactique. *Petit x*. 83. 27-49

PAPERT, S. (1981) *Jaillissement de l'esprit, ordinateurs et apprentissages* (texte français de R.M. Vassallo-Villaneau). Paris : Flammarion,

ROGALSKI, J. (1988) Les représentations mentales du dispositif informatique dans l'alphabétisation. In C. Laborde (Ed.), *Actes du 1er colloque franco-allemand de didactique des mathématiques et de l'informatique* (pp. 237-245). Grenoble : La Pensée Sauvage.

ROGALSKI, J. (2015) Psychologie de la programmation, didactique de l'informatique : déjà une histoire… In G.-L. Baron, E. Bruillard, E., & B. Drot-Delange (Eds.), *L'information en éducation : perspectives curriculaires et didactiques* (pp. 279-305). Clermont-Ferrand : Presses Universitaires Blaise Pascal.

ROGALSKI, J., & HE, Y. (1989) Logic abilities and mental representations of the informatical device in acquisition of conditional structures by 15-16 year-old students. *European Journal of Psychology of Education*, 4(1), 71-82.

ROGALSKI, M., SAMURÇAY, R. (1990) Acquisition of programming knowledge and skills. In J. M. Hoc, T. R. G. Green, R. Samurcay, D. J. Gilmore (Eds) *Psychology of programming* (pp. 157-173). London: Academic Press.

ROUCHIER, A. (1990) Objets de savoir de nature informatique dans l'enseignement secondaire. *ASTER, 11, Informatique, regards didactiques*, 29-44.

RUTHVEN, K. (1996) Calculators in the mathematics curriculum: The scope of personal computational technology. In A. Bishop, K. Clements, C. Keitel, J. Kilpatrick & C. Laborde (Eds.) International Handbook of Mathematics Education (pp. 435-468). Dordrecht: Kluwer.

SAJANIEMI, J. (Ed.) (2008) Special issue on Psychology of Programming. *Human Technology*, 4(1).

SAMURÇAY, R. (1985) Signification et fonctionnement du concept de variable informatique chez des élèves débutants. *Educational Studies in Mathematics*, 16(2), 143-161.

SAMURÇAY, R. (1989) The concept of variable in programming: its meaning and use in problem-solving by novice programmers. In E. Soloway & Spohrer, (Eds.), *Studying the novice programmer*. Hillsdale, NJ : Lawrence Erlbaum Associates.

SAMURÇAY, R. (1989) The concept of variable in programming : Its meaning and use in problem solving by novice programmers. In E. Soloway & J.-C. Spoher (Eds.), *Studying the novice programmer* (pp. 161-178). Hillsdale, NJ : Lawrence Erlbaum Associates.

SAMURÇAY, R., & ROUCHIER, A. (1990) Apprentissage de l'écriture et de l'interprétation des procédures récursives. *Recherches en didactique des Mathématiques*, 10 (2-3), 287-327.

SORVA, J. (2008) A Roles-Based Approach to Variable-Oriented Programming. Human Technology, Volume 4 (1), pp. 62-74.

TALL, D.O., & THOMAS, M.O.J. (1986) The value of the computer in learning algebra concepts. *Proceedings of the 10th Conference of PME* (pp. 313-318). London : University of London Institute of Education.