informatique

alphabétisation informatique

Problèmes conceptuels et didactique

par Janine Rogalski
chargée de recherches au CNRS

Tout enseignement dans une discipline confronte enseignants et élèves à des difficultés conceptuelles ; ce qui touche à l'informatique n'échappe pas à cette règle commune. Nous voudrions, à propos de ces questions, susciter le débat sur les acquisitions conceptuelles des élèves, sur les moyens de les étudier et d'en repérer d'éventuels obstacles, sur les problèmes d'enseignement de la partie spécifique de l'informatique qu'est la programmation. Pour cela nous nous appuyerons sur les observations conduites pendant deux ans lors d'un enseignement élémentaire de la programmation à des élèves de seconde (1).

Nos analyses nous conduisent à mettre en avant quatre points :

— une "informatique pédagogique existe, au sens où une discipline particulière est devenue un objet d'enseignement (à visée non professionnelle) à un titre proche de celui de la physique ou des mathématiques dans l'enseignement général ;

— les notions nécessaires à l'alphabétisation informatique présentent des difficultés conceptuelles propres dont l'enseignement doit tenir compte et sur lesquelles la recherche doit se développer ;

(1) Une expérience d'enseignement des premiers éléments de la programmation a été conduite en classe de seconde, dans le cadre d'un contrat ADI (projet EAO 696 ABC. Convention 81/171.172.173.) sous la responsabilité de A. Rouchier et G. Vergnaud, avec J. Rogalski, R. Samurçay, J.-C. Despland, J.-M. Laubin, Y. Ferrand, C. Landré, J.-F. Pigeonnat, G. Sarfati. Cf. Rapport de recherche, Rouchier et al, 1984.

— la conception que l'on a de l'initiation à la programmation produit des effets sur les représentations des élèves quant à l'informatique ; elle a des conséquences également sur les choix que les enseignants doivent maîtriser dans le déroulement de leur enseignement ;

— le développement de la didactique en informatique devient important à double titre : pour l'enseignement de l'informatique elle-même et pour une meilleure maîtrise des acquisitions des élèves en mathématique dans la perspective d'une intervention significative de l'informatique dans cet enseignement (*).

1. L'existence d'un processus de "didactification" de l'informatique.

L'informatique n'est ni une technologie, ni un appendice de la logique ou des mathématiques : en quelques décennies (2) une discipline s'est organisée, avec un ensemble complexe de concepts, de représentations, de problèmes. Dans son enseignement se manifeste le phénomène analysé — pour les mathématiques d'abord — par Chevallard (1980) sous le nom de *transposition didactique*, à savoir la transformation qui fait passer de l'objet de connaissance sur lequel travaillent des "professionnels" à un objet d'enseignement, pour un public d'enseignés plus ou moins explicitement déterminé. Il suffit pour se convaincre de l'existence de cette "didactification" de comparer par exemple la présentation d'un langage comme PROLOG dans une revue spécialisée et l'introduction à ce langage dans un document à visée pédagogique.

Une autre manifestation de cette transposition est le développement de manuels qui précisent le public auxquels ils s'adressent et les prérequis éventuellement nécessaires (3). L'introduction en 1981, à titre expérimental, d'un enseignement optionnel d'informatique dans le second cycle avec un programme cadre, contribue à cette didactification de l'informatique (4). A côté des documents propres à différents groupes d'ensei-

(*) Rapport pour la CIEM (Bulletin 345, page 509), article de L. SOLOMON (Bulletin 343, page 231).

(2) La "pose de jalons théoriques" peut être datée des années 30 de ce siècle (Turing, 1936) ; la création du premier langage de large diffusion : FORTRAN date des années 50 (1954-1957) ; l'intérêt apporté à la méthodologie de la programmation remonte aux années 65-70 (Djikstra, 1968). Pour un panorama historique on peut par exemple se reporter à /Meyer et Baudoin, 1980/.

(3) Pour le domaine de la programmation on peut citer comme exemples : /Arsac 1977/, /Baudoin et Meyer 1980/, /Lucas et al 1983/, /Horowitz 1983/. Pour l'enseignement secondaire, à côté des documents propres à divers groupes d'enseignants existe un premier manuel /Arsac-Mondou et al 1983/. Par ailleurs la succession des préambules aux éditions de Crocus /1975, 1976, 1981/ donne des éléments d'interrogation sur l'interaction entre l'évolution technologique et celle de la didactique concernée.

(4) L'objectif des initiateurs de cette expérience n'est pas l'introduction de l'informatique comme une discipline nouvelle s'ajoutant aux autres. Cependant la nécessité est reconnue par tous d'assurer des bases minimales pour utiliser l'informatique comme "auxiliaire de pensée" dans d'autres disciplines : cela se concrétise par une "didactification" des notions élémentaires de programmation.

gnants, la sortie d'un premier manuel (Arsac-Mondou et al, 1983) marque une reconnaissance institutionnelle (par le système éditorial concerné par l'enseignement secondaire) de l'informatique comme "discipline enseignée".

Certes un débat est engagé sur le statut de l'introduction de l'informatique dans l'enseignement : notre thèse est que, d'une part, même si la perspective est centrée sur l'informatique comme "auxiliaire de pensée", selon la formule de C. Pair (s'exprimant en tant que Directeur des lycées, à propos de l'option informatique dans le second cycle), on ne peut échapper à la question des représentations que se font les élèves des concepts informatiques et à la question du fonctionnement cognitif dans ce domaine de connaissances, et que d'autre part — ne serait-ce que dans la perspective précédente — l'alphabétisation informatique devient tout aussi nécessaire que l'initiation aux mathématiques pour tous ceux dont ce ne sera jamais la profession.

L'étude de l'acquisition des concepts (élémentaires) propres à l'informatique et l'analyse de leur enseignement présentent donc une importance non seulement du point de vue de la connaissance didactique mais du point de vue de la pratique pédagogique dans diverses disciplines et des implications sociales de l'informatique. La communauté informatique, excessivement sollicitée par ailleurs, a encore relativement peu investi dans l'épistémologie et la didactique de sa discipline, mais les acquis théoriques et méthodologiques de la didactique des mathématiques peuvent être efficaces dans une collaboration qui devrait se développer.

2. Les difficultés conceptuelles dans l'acquisition des premières notions informatiques sur la programmation : l'exemple des structures itératives.

Les travaux diffusés jusqu'ici sur les premières acquisitions concernent essentiellement des étudiants plus engagés dans leur cursus scolaire que les élèves de l'enseignement secondaire et le problème de la validité des méthodes et des résultats pour les élèves plus jeunes doit être posé. Certes, des expériences maintenant assez nombreuses d'introduction de la programmation "tôt" dans la scolarité ont dessiné les contours de ce que l'on pouvait faire avec des élèves jeunes ; toutefois ces expériences sont souvent présentées du point de vue de ce que l'enseignant a fait et apportent des informations ponctuelles sur les acquisitions des élèves et sur la gestion des contenus en jeu au cours du déroulement de la classe(5).

Nous allons présenter quelques éléments sur un point particulier — mais central — dans les acquisitions en programmation : les structures de répétition, étudiées dans des langages "procéduraux".

(5) L'essentiel de ces expériences concerne l'introduction de LOGO avec la géométrie de la tortue. L'interaction de l'activité de la programmation avec la nature de l'objet sur lequel porte cette activité a été soulignée dans un certain nombre de travaux, voir en particulier /Rouchier 1981, 1982, 1983/.

Il y a des méthodes multiples pour étudier cette question des difficultés conceptuelles et des représentations des élèves sur les notions en jeu.

On peut étudier la façon dont les élèves expriment spontanément les instructions nécessaires pour résoudre un problème algorithmique qu'ils savent par ailleurs résoudre "à la main", dans une situation où ces instructions doivent être exécutées par un autre opérateur qui ne connaît pas a priori le problème auquel elles répondent (6).

On peut analyser de façon "clinique" le processus de résolution d'un problème informatique, sous la forme d'"étude de cas" soit d'élèves individuels, soit de groupes d'élèves (ce qui permet d'avoir parmi les observables des verbalisations spontanées).

On peut organiser une interrogation plus standardisée, par exemple de programmes (ou d'algorithmes) à compléter, faire "tourner à la main", transformer, etc...

Au niveau de l'alphabétisation, la construction d'algorithmes complets (et/ou de programmes) est souvent peu informative pour la recherche en didactique dans la mesure où les différentes difficultés sont imbriquées les unes aux autres : elle est en revanche décisive si on veut analyser la planification d'un ensemble significatif d'opérations (c'est-à-dire en comportant à la fois un nombre suffisant et représentant un champ assez large dans les différents types d'opérations possibles).

Globalement, les observations effectuées sur une séquence d'introduction à la programmation en classe de seconde (cf (1)) convergent avec des études comme celles de Soloway et Ehrlich sur des "débutants" et sur des étudiants plus avancés : les structures de répétition ne sont ni simples ni équivalentes en difficulté.

Ainsi, par exemple, les représentations spontanées sur l'expression de la répétition nécessaire au calcul par additions successives ($n+n+..+n$, p fois) du produit np de deux entiers positifs ne respectent tout d'abord pas nécessairement la séquentialité, et présentent ensuite les caractères suivants :

— les opérations à effectuer sont décrites avant l'indication de la répétition ;

— la répétition est du type "répète" (ou "recommence").. "jusqu'à ce que"..., la condition d'arrêt étant exprimée en dernier lieu ;

— à l'intérieur de la boucle, l'opération significative — c'est-à-dire celle dont la succession des exécutions va produire le résultat recherché — est présentée avant celle qui contrôle le nombre d'exécutions.

L'analyse des productions de groupes d'élèves (2 à 3 élèves par groupe) dans l'écriture de l'algorithme permettant d'obtenir la somme des

(6) Le terme de "problème conceptuel" utilisé ne doit pas conduire à l'idée qu'on devrait rechercher des moyens pédagogiques de les contourner ou de les éliminer : à notre sens c'est bien l'existence de problèmes conceptuels qui justifie l'existence d'un enseignement dans le domaine considéré.

n premiers entiers (*n* étant choisi par l'opérateur) a permis d'étudier les différentes phases dans la solution de ce problème; elle a mis en évidence des décalages importants entre les groupes étudiés, cela après seulement quelques heures d'enseignement; elle a confirmé le statut particulier de l'instruction du corps de boucle qui incrémente la variable "compteur" (Rouchier et al.1984, Samurçay 1984, Rogalski 1984).

L'hypothèse générale que nous avons faite est que les difficultés conceptuelles des élèves dans des tâches de programmation seraient d'autant plus grandes que la structure de la solution informatique s'éloignerait des représentations spontanées et des méthodes de solution "à la main"; cette hypothèse conduit aux conséquences suivantes, en ce qui concerne les structures de boucle:

— la structure *"répète/*action*/jusqu'à/*condition*/"* est la plus accessible;

— à l'intérieur de cette boucle, l'explicitation de la transformation d'une variable "compteur", qui contrôle le nombre d'exécutions, ne pose pas de problème majeur (dès lors bien entendu que la séquentialité est respectée);

— le test sur cette variable "compteur" reflète les représentations spontanées et est donc sans problème;

— l'inversion des opérations à l'intérieur du corps de boucle nécessite une construction cognitive relativement difficile;

— l'inversion des opérations d'action et de test pour la structure de répétition *"tant que/*condition*/faire/*action*/"* exige de mettre en cause le modèle spontané, on doit donc rencontrer des obstacles importants dans son acquisition;

— la structure du type *"pour i* = 1 à *n faire"*, dans laquelle la variation de la variable "compteur" (*i*) ainsi que son test ne sont pas explicites, va fonctionner d'abord comme une instruction "miracle" pour laquelle la question de l'adéquation au problème à résoudre aura très difficilement du sens.

Ces éléments sont compatibles d'une part avec les analyses faites de séances ultérieures à la séquence didactique (Rouchier, Samurçay et al, 1984) d'autre part avec les résultats obtenus par Soloway (Soloway, Ehrlich et al, 1982a) sur des étudiants; cet auteur constate en effet que la structure "tant que..." pose encore des problèmes aux étudiants "avancés" et que les "étudiants" ont des difficultés dans la compréhension de la boucle "pour..." qui, selon ses hypothèses aurait dû être plus facile puisque contrôlée par le processeur.

Nous avons par ailleurs construit et analysé une séquence didactique destinée à introduire la structure "tant que..." comme solution appropriée à certains types de problèmes, à partir de l'utilisation de programmes produits en utilisant la structure antérieurement apprise: "répète", et éventuellement la structure conditionnelle ("si...alors...sinon"). Les solutions initiales apportées par les élèves utilisent deux fois un test portant sur la même variable booléenne (dans la condition du

"répète...jusqu'à" et dans le "si...alors...sinon") dont la coordination leur pose des problèmes.

Les différences importantes observées entre élèves semblent en rapport avec les acquisitions d'ensemble des premiers concepts de programmation, et plus précisément, avec la compréhension des relations entre l'unité d'initialisation de la boucle et la modification des variables dans le corps de boucle.

3. Niveaux des problèmes conceptuels dans la programmation informatique et questions didactiques

Si nous reprenons la formulation utilisée pour présenter les objectifs de l'enseignement (option expérimentale) de l'informatique dans le second cycle du secondaire, nous pouvons dégager quatre plans distincts (bien que non indépendants) dans lesquels se posent des problèmes d'acquisitions conceptuelles (6). (Nous avons séparé les différents objectifs explicités et souligné les points que nous analyserons brièvement.)

" ...permettre aux élèves de lire un texte pour en dégager une *formulation précise du problème à résoudre*
" en trouver une *méthode de résolution*
" *la rédiger dans un langage* de programmation
" et la faire *exécuter par un ordinateur*."

Cette représentation qui fait apparaître 4 phases distincts dans un problème de programmation informatique nous paraît distinguer fortement une telle situation de celles rencontrées par ailleurs en mathématiques (ou dans d'autres situations de résolution de problème).

3.1. Tout d'abord la définition du problème à résoudre fait ici partie de la tâche, alors que — par exemple — dans un problème mathématique du contexte scolaire, le contrat considère au contraire le problème comme parfaitement déterminé : c'est à l'enseignant qu'est dévolue cette définition du problème et non à l'élève (sinon le problème sera dit "mal posé"). Le changement de contrat est drastique puisque la charge de définir précisément le problème passe de l'enseignant à l'élève.

3.2. Ensuite la recherche d'une méthode de résolution se distingue de la résolution "à la main" : d'une part, du fait du passage d'un opérateur humain à un processeur, les algorithmes peuvent mettre en œuvre des opérations de natures très différentes ; d'autre part, le caractère de validité générale d'une méthode de résolution par rapport à une solution "locale" fait changer de niveau de résolution.

3.3. Par ailleurs la méthode de résolution est recherchée avec une orientation particulière : il s'agira en effet de la rédiger ensuite dans un langage de programmation. Les caractéristiques de ce langage, les rapports entre la structuration des données et celle des instructions, les con-

traintes syntaxiques doivent être un objet d'enseignement ; les observations faites sur des langages comme BASIC, LOGO, PASCAL, indiquent qu'au-delà des questions de syntaxe, les problèmes de signification jouent un rôle très important. (Par exemple les confusions entre les instructions de lecture et d'écriture — qui traduisent des opérations d'entrée et sortie — observées en LSE ou PASCAL semblent ne pas se produire lorsque le langage les présente explicitement comme des procédures — comme en LOGO —).

3.4. Enfin, l'exécution de la solution écrite dans un langage déterminé pose des problèmes de divers ordres, qui touchent à l'appropriation d'un "objet matériel fabriqué" particulier (7). La représentation des opérations en jeu pour "entrer un programme" sont en interaction avec les propriétés du système ; celles qui concernent l'exécution elle-même touchent aux rapports entre l'ordinateur, le programme et l'opérateur (qui le "fait tourner"), en particulier la représentation des entrées-sorties et de la gestion de l'écran n'est pas immédiate (8). (On aborde ici, quoique tangentiellement, un autre domaine de l'informatique : celui des systèmes.)

Au niveau de l'alphabétisation il semble, au vu d'un certain nombre d'observations ponctuelles, que l'enseignement de la programmation soit centré essentiellement sur deux phases : "trouver une méthode de résolution" et "la rédiger dans un langage de programmation". De plus, une évolution se dessine dans le sens d'une distinction très forte des deux objets : *l'algorithme* et son *écriture* (dans un langage de programmation) ; l'accent est alors mis sur l'algorithme, des règles de traduction sont fournies ensuite. Les raisons de cette évolution sont explicites : il s'agit d'apprendre aux élèves des méthodes (relativement) indépendantes d'un langage particulier et qui résisteront aux changement d'un langage à un autre et aux évolutions dans l'accessibilité des langages.

Existence d'"effets de bord" produits par la marginalisation des phases de définition du problème et d'exécution effective.

Nous voudrions donner des exemples d'"effets de bord" produits par la centration sur les phases centrales : il s'agit d'effets qui ne relèvent pas de l'objectif "algorithme" visé, et qui en l'occurrence peuvent être

(7) Une équipe de l'INRP, conduite par P. Rabardel, en collaboration avec F. Léonard, a initié un travail sur le thème "objets matériels fabriqués et développement cognitif". Ils définissent un objet matériel fabriqué comme une *unité matérielle* identifiable, *produit* d'une activité humaine *finalisée* dans un certain but. Le micro-ordinateur répond à l'évidence à cette définition. Un tel "objet matériel fabriqué" (OMF) est "porteur d'une quantité importante de connaissances" dont l'appropriation par un sujet constitue un objet d'étude dans le champ du développement socio-cognitif.

(8) Pour prendre un exemple flagrant : le double statut de l'écran comme "écho" à ce que l'opérateur entre au clavier et comme interface de sortie pose des problèmes au débutant, problèmes surtout sensibles si l'exécution du programme est considérée par l'élève comme l'élément de validation du programme. Pour certains types de problèmes des propriétés temporelles de la visualisation ne sont pas toujours faciles à maîtriser rapidement.

considérés comme non souhaitables. Prenons un exemple simple : supposons que l'objet en jeu soit l'itération et plus précisément la structure "pour..." ; supposons que le problème choisi pour la mettre en œuvre soit le suivant : écrire l'algorithme (ou le programme) qui donne les carrés des nombres entiers successifs, le dernier nombre étant choisi par l'opérateur. Il faut souligner que si l'enseignant veut centrer l'objectif sur la structure "pour..." il doit préciser fortement le problème (cf. infra la situation inverse).

Un algorithme répondant à cet objectif est par exemple le suivant :

entrer(n)
pour $i = 1$ à n faire
    carré = $i * i$
fin-pour

Mais une fois l'algorithme écrit, qui doit en éprouver la validité ? et par quels moyens ? Si la validation est faite par une "exécution à la main", ce qui est un mode fréquent chez les élèves, elle va nécessairement concerner de petites valeurs de n (choisies entières, strictement plus grandes que 1). Du point de vue algorithmique, il manque la protection contre des entrées illicites. L'élève passe à la traduction, par exemple en Pascal :

```
programme carrés ;
var  i , carre : integer ;
begin
    readln(n) ;
    for i := 1 to n do carre := i*i ;
end.
```

Malgré la déclaration des variables, le programme n'est pas protégé contre des entrées illicites puisque les entiers sont ici dès relatifs (il faut souligner que certaines entrées illicites peuvent être bloquées — au prix du blocage du programme — à partir de la déclaration du type des variables imposée par Pascal, mais que le contrôle de type peut ne pas avoir lieu lors de la lecture...) (9).

Lors de l'exécution :
temps 1 : rien n'apparaît sur l'écran ;
temps 2 : l'élève se souvient qu'il faut entrer un entier n ; il tape un nombre, qui s'inscrit sur l'écran (par "écho") ;
temps 3 : rien ne se passe / sur l'écran / ; réaction (entendue souvent dans ce cas) : "ça ne marche pas" ; la logique de l'algorithme validé plus tôt peut être mise en cause par une exécution où l'élève ne voit rien se passer : centrée sur l'algorithme, la définition du problème était muette sur les sorties : nous considérons qu'il y a là un premier "effet de bord".

(9) Nous signalons que nous avons laissée ouverte la question des protections contre des entrées illicites de la part de l'opérateur (erreurs de frappe, nombres non entiers, non compris entre 1 et 150, réponse autre que O ou N..). Comme le font souvent spontanément les élèves, on peut faire un premier choix qui laisse l'opérateur encourir ces risques d'erreurs.

Réciproquement, des définitions du problème qui prendraient systématiquement en charge la question des affichages pour obtenir la cohérence entre algorithme et programme souhaité peuvent conduire l'élève à considérer qu'un programme ne fait quelque chose que lorsqu'il y a un affichage ; cela peut aussi le conduire à faire gérer la validation de ses algorithmes par l'exécution du programme sur l'ordinateur : deuxième effet de bord. Il faut d'ailleurs souligner que les sorties réelles sur écran, qui font partie de la dernière des phases présentées plus haut (ce sont les plus fréquentes, quand elles ne sont pas exclusives) posent en retour d'autres problèmes de programmation.

Si on n'aborde pas ce problème des sorties, et de leur réalisation concrète sur le matériel avec lequel on travaille, on fait disparaître un élément décisif d'un programme informatique : il est destiné à un utilisateur, fût-il son concepteur lui-même !, et sa conception doit prendre en compte ce fait, sinon on contribue à ce que l'élève ait une représentation de la programmation réduite à l'algorithme et étrangère à l'utilisation de l'informatique (par celui qui se servira du programme)(10) : c'est un troisième "effet de bord" d'un contrat centré sur une partie seulement des 4 phases identifiées comme objectifs de l'enseignement de l'informatique.

Quelques problèmes didactiques à résoudre dans la prise en charge des diverses phases de la programmation informatique

Nous avons indiqué ci-dessus quelques exemples de "biais" de représentations des objets de la programmation par une centration exclusive sur les algorithmes et leur "traduction" dans un langage.

Nous allons maintenant donner un exemple de situation dans laquelle, si on considère seulement le problème algorithmique dans un sens "mathématique", on peut se trouver dans une situation difficilement maîtrisable où les questions à résoudre sortent éventuellement du champ des acquisitions dans lequel on veut faire travailler les élèves.

Nous allons voir sur cet exemple simple que, selon la situation didactique que l'enseignant veut produire : choix d'une structure particulière ou comparaison de différentes solutions, il doit faire des choix tout à fait différents par rapport à la définition du problème et aux propositions éventuelles des élèves.

Prenons le problème "imprécis" suivant : faire un programme qui permette d'avoir le carré de tout nombre entier entre 1 et 150 (cette précision numérique évite d'avoir à prendre en compte le problème de représentation des entiers). Supposons de plus que "avoir" un résultat signifie

(10) De nombreux rapports ont mis en évidence les problèmes de documentation et les relations avec le caractère évolutif des logiciels : c'est souvent le point de faiblesse des logiciels. Que les élèves doivent plus tard intervenir à quelque niveau que ce soit dans la constitution de tels objets ou qu'ils n'en soient que des utilisateurs, la prise de conscience de l'importance de la documentation est en tout état de cause un objectif de l'enseignement.

"pouvoir lire sur l'écran" : on peut penser que c'est une précision qui viendrait facilement du débat en classe ou des solutions individuelles ; à la limite l'enseignant peut compter sur des phénomènes de diffusion dans la classe d'une telle interprétation (vu les équipements actuels).

Il reste deux grandes options ouvertes, qui vont conduire à des structures de répétition très différentes :
— "tout nombre" signifie "tous les nombres", le programme devra afficher le carré de tous les nombres entre 1 et 150 ;
— "tout nombre" signifie "quel que soit le nombre", le programme affichera le carré de n'importe quel nombre, au choix de l'opérateur.

La première option correspond à la constitution d'une table des carrés, la seconde option à la donnée immédiate d'un carré quand on donne un nombre.

La réalisation de la première option fait raisonnablement appel à la structure de boucle "pour...", mais pour être opératoire elle exige de gérer les problèmes d'affichage : l'écran a un nombre de lignes inférieur à la borne choisie pour n, et l'affichage de chaque ligne est trop bref pour la lecture : le programme doit donc gérer le temps ou l'espace de l'affichage en découpant l'intervalle numérique [1 ; 150] en fonction du nombre de lignes utilisées de l'écran. Cela conduit à des problèmes complexes pour des élèves du secondaire (voire au-delà...) de calcul modulo p, et/ou à des problèmes complexes de programmation d'une double répétition "pour..." en introduisant la succession des affichages de chaque "bloc" lisible sur l'écran.

L'"effet de bord" que nous avions présenté dans la partie précédente est évité au prix d'un changement notable de niveau de complexité : le choix didactique ne peut donc guère concerner une seule séance de cours et sa maîtrise n'est à l'évidence pas une question simple...

La seconde interprétation implique la programmation soit d'une boucle sans fin : le programme fournit à la demande un carré, soit d'une boucle contrôlée par la valeur d'un choix de l'opérateur sur la poursuite (voire éventuellement le démarrage) du travail. Selon l'option prise dans la détermination du problème les structures d'itération appropriées ne sont pas de même difficulté :

Test d'achèvement du travail : le programme lit un nombre, affiche son carré jusqu'à ce que l'opérateur décide de s'arrêter ; la structure appropriée est une structure "répète..."

```
répète
    lire(n)
    afficher(n*n)
    afficher('avez-vous terminé ?O/N')
    lire(rep)
jusqu'à rep = 'N'
```

Test de démarrage : le programme demande si l'opérateur désire le carré d'un nombre ; tant que la réponse est positive il lit le nombre, affiche son carré, repose la question : la structure appropriée est une structure "tant que...faire..." ; par exemple :

```
afficher('voulez-vous le carré d'un nombre ?O/N')
lire(rep)
tant que rep = O faire
    lire(n)
    afficher(n*n)
    afficher('voulez-vous le carré d'un autre nombre ?O/N')
    lire(rep)
fin-tant que
```

Utilisation illimitée du programme : une fois que le programme est lancé l'opérateur ne se préoccupe que de rentrer des nombres. On peut utiliser l'une des deux structures précédentes avec une condition toujours vraie ou toujours fausse : la difficulté cognitive n'est pas la même selon le cas (la manipulation d'hypothèses fausses apparaît difficile dans un certain nombre de raisonnements). Quant à l'utilisation du GOTO ou celle de la récursion infinie, elles posent d'autres problèmes (11).

Dans les choix de définition du problème à traiter, les décisions n'apparaissent donc pas simples.
— pour contrôler la structure de boucle que l'on veut faire utiliser par la classe, on peut "forcer" le choix des élèves : cela peut poser des problèmes de contrat avec les élèves et même cela peut vider de sens le travail sur la définition du problème :
— si au contraire, l'enseignant veut obtenir une variété de solutions correspondant à des choix de problèmes distincts, il risque de ne pas les obtenir, même en travaillant avec un nombre important d'élèves : en effet les solutions sont de niveaux cognitifs différents, et la probabilité est grande que seuls certains problèmes soient proposés.

4. Conclusion

Nous avons explicité brièvement quelques points touchant aux difficultés conceptuelles et aux problèmes didactiques rencontrés dans l'enseignement de la programmation informatique, cela en prenant l'exemple de l'itération.

Les éléments apportés actuellement par la recherche confirment que les acquisitions dans ce domaine posent des problèmes cognitifs importants aux élèves (lorsqu'il s'agit de classes sans caractéristiques particuliè-

(11) L'utilisation de types de boucles infinies par un GOTO ou une récursion ne pose pas toujours de problèmes apparents aux élèves mais nous pensons que cette absence de problème tient justement au fait qu'ils n'ont pas à gérer dans ces boucles l'objet difficile : à savoir l'état d'une certaine variable booléenne. D'un point de vue didactique, même si le langage choisi permet une telle solution, son utilisation nous paraît être "un coup pour rien".

res), et qu'il existe des difficultés spécifiques tenant au passage de la résolution de problèmes "à la main" à un programme informatique.

Par ailleurs la question du choix de ce qui doit être entendu sous le titre d'"initiation à la programmation et/ou à l'informatique" n'est pas sans effets, d'une part sur le plan des représentations des élèves quant à l'informatique, d'autre part sur le type de problèmes posés et sur les choix didactiques des enseignants.

Les éléments de discussion, très partiels, que nous souhaitons avoir introduits ici, ont été tournés vers l'objet "informatique", avec l'hypothèse, que nous avons explicitée, que l'usage de l'informatique dans la transformation de l'enseignement d'une autre discipline — comme les mathématiques — nécessitait une alphabétisation réelle en informatique.

Par ailleurs le changement de l'enseignement des mathématiques, et de son contenu, en liaison avec l'introduction de l'informatique, nous semble nécessiter aussi un important développement des recherches sur les concepts informatiques comme ceux de variables, constantes, conditions, itération et récursivité, procédures, représentations des nombres... en relation avec les concepts mathématiques qui interagissent avec eux ; cela nous paraît une condition nécessaire (bien que non suffisante) pour avoir une certaine maîtrise des objets mathématiques qui se construiront "dans la tête" des élèves à partir d'une intervention importante de l'informatique dans leur enseignement.

Bibliographie

ANDERSON J.R., FARELL R., SAUERS R. (1983). Learning to programming in LISP. *Report of the department of psychology*. Carnegie-Mellon University, Pittsburg, PA-15213.

ARSAC J. (1977). *La construction de programmes structurés*, Dunod, Paris.

ARSAC J. (1980). *Premières leçons de programmation*, Cedic/Fernand Nathan, Paris.

ARSAC-MONDOU O., BOURGEOIS-CAMESCASSE C., GOURTAY M.
(1982) *Premier livre de programmation*.
(1983) *Deuxième livre de programmation*.
(1983) *Pour aller plus loin en programmation*. Cedic - Fernand Nathan, Paris.

BAUDOIN C., MEYER B. (1978). *Méthodes de programmation*, Eyrolles, Paris.

BERDONNEAU C., BOSSUET G. (1983). Quelques exemples de comportements d'enfants de l'école primaire dans un environnement LOGO. in *L'informatisation dans l'éducation scientifique*, Rapport des 4° journées sur l'éducation scientifique, 111-119.

COLLOQUE LOGO (1°) (1983). *Enseignement, formation et pratique active de l'informatique par l'enfant.* INRP, IREM, Clermont-Ferrand - Orléans.

CROCUS (1975-1976-1981) *Systèmes d'exploitation des ordinateurs.* Dunod-informatique, Paris.

DJIKSTRA E. (1968) A constructive approach of the problem of program correctness ; BM 8, 174-186.

DJIKSTRA E. (1971) A short introduction to the art of programming ; report EWD 316. Technische Hogeschool Eindhoven.

DJIKSTRA E. (1976) *A discipline of programming,* Prentice Hall.

ENNALS R., BOULCEMA O., BERGMAN M. (1983). *The french connection.* The Commodore Educational Computing Conference, Londres.

ENNALS R. (1982). *Teaching logic as a computer language in schools.* First International Logic Programming Conference, Marseille.

HOC J.M. (1979) *Le problème de la planification dans la construction d'un programme informatique.* Le travail humain, 42, (2), 245-260.

HOC J.M. (1983) *La programmation informatique à l'école : les exigences de cette activité.* Rapport sur les 4° journées sur l'éducation scientifique, Chamonix.

KOWASKI R. (1982). *Logic as a computer language for children.* ECAI, Orsay, France.

MEJIAS B. (1983) *Etude de la notion d'itération au moment de la mise en place et de l'expression d'un algorithme avec boucle.* Seminaire national de didactique des mathématiques, Paris.

ROGALSKI J. (1983) *Etude de cas dans une situation d'élaboration de procédure informatique.* Journées sur la résolution de problèmes, La Baume.

ROGALSKI J. (1983) Sur une séquence didactique en informatique. *Séminaire de didactique et pédagogie des mathématiques* IMAG, Grenoble.

ROUCHIER A. (1981). *Problèmes, procédures, programmes étudiés et réalisés par des enfants de CM2 utilisant un mini-ordinateur.* Revue française de pédagogie, n° 56, 18-26.

ROUCHIER A. (1982) *Sa Majesté la Tortue.* Education et informatique. Vol. I0, p. 23-24.

ROUCHIER A. (1983) Actes du premier rapport LOGO. INRP ; Clermont/Orléans, IREM Orléans.

ROUCHIER A., SAMURÇAY R., ROGALSKI J. (1983). *Analyse d'une séquence didactique sur la programmation informatique,* Séminaire national de didactique des mathématiques, Paris.

ROUCHIER A., SAMURÇAY R., ROGALSKI J., VERGNAUD G., DESPLAND J.C., et coll. (1984) *Concepts informatiques et programmation. Une première analyse en classe de seconde des lycées.* CEPCL, Paris, IREM Orléans.

SAMURÇAY R. (1983a) *Informatique et didactique des concepts informatiques*. Troisième rapport intermédiaire de contrat : IREM Orléans.

SAMURÇAY R. (1983b) *Quelques aspects du concept de variable dans la programmation informatique*. Journées sur la résolution de problème, La Baume.

SAMURÇAY R. (1984b) *Quel langage pour apprendre ? Aujourd'hui le PASCAL*. Led Micro, 9, 40-41.

SAMURÇAY R. (1984a) *Problèmes cognitifs posés par l'acquisition des concepts informatiques : variable et boucle conditionnelle*. Centre d'étude des processus cognitifs et du langage. CNRS-EHESS, Paris.

SOLOWAY E., EHRLICH K., BONAR J., GREESPAN J. (1982). *What do novices know about programming*. Research report, Yale-University.