INFORMATIQUE ET APPRENTISSAGE

*François BOULE**

L'informatique a de multiples raisons ou occasions d'intervenir dans le système éducatif. La découverte de l'informatique comme phénomène global et vecteur de transformation sociale, l' ordinateur comme objet technique, éventuellement extensible par des robots, l'utilisation de l'ordinateur à des fins de gestion de données ne seront pas envisagées ici. L'enfant à l'école apprend à être et à penser en acquérant des connaissances. En quoi l'informatique peut-elle contribuer à cette tâche ?

I – ENSEIGNEMENT ASSISTE PAR ORDINATEUR (EAO)

On a pris coutume d'opposer les conceptions qui ont donné naissance d'une part à l'EAO,d'autre part au système Logo. L'opposition a été ainsi résumée par S. Papert : dans le premier cas c'est la machine qui programme l'enfant, dans le second l'enfant qui programme la machine. L'enseignement programmé (programmeur ?) paraissait s'inspirer du béhaviorisme et visait à renforcer des comportements. Les créateurs de Logo se réclament d'une psychologie construtiviste : l'enfant construit (et réorganise) son propre savoir. La réalité est probablement moins manichéenne, mais demeure difficile à analyser.

Il convient de remarquer que lorsqu'on le fait fonctionner,un ordinateur n'est jamais privé de logiciel (soft) à un niveau ou à un autre : en Rom**, ou bien sous forme de langage disponible, ou encore d'un programme particulier. Selon ces cas, c'est l'espace de ses comportements possibles qui est plus ou moins réduit ; faute du moindre programme (câblé en Rom), il est inerte ; bridé par le lancement d'un programme non interactif, il est orienté sans déviation possible (sauf panne) vers le point d'achèvement. *Entre ces deux extrêmes,* il peut produire des résultats plus ou moins divers et nombreux : l'éventail s'élargit.

Et du côté de l'enfant ? Si l'enfant répond à un QCM, son espace d'évolution est réduit : agir quand il est sollicité et dans les étroites limites fixées. Il se peut néanmoins que le programme tienne compte de ses réponses,c'est-à-dire qu'il y ait plusieurs trajectoires possibles. Mais il n'en a pas nécessairement conscience. A l'inverse si l'enfant dispose d'un *langage de programmation,* il peut faire ce qu'il veut, dans les limites toutefois qu'il **sait** ou qu'il **croit**

** – Cet article est un compte rendu de conférences, déjà paru dans "Les cahiers de Beaumont" en mars 1986.*

*** – Rom : Read only memory.*

être celles du langage ; un langage comporte un certain nombre de règles d'usage restrictives ; il s'identifie à certains modes de représentations (lesquelles dépendent de la connaissance que le programmeur a du langage). C'est donc en terme de *degré de liberté* que l'on peut imaginer l'analyse de l'interaction.

Mais il y a bien d'autres directions d'analyse :

a) le rôle de l'adulte

L'EAO "classique" (c'est une figure historique et quasi-fictive, mais un enjeu politique et polémique) permet un fonctionnement autonome ; l'acte d'enseigner est dispensé par la machine et dispense du maître. Selon une certaine conception (initiale) de Logo (cf. S. Papert, G. Bossuet, etc.) l'adulte est personne-ressource, recours éventuel, mais désaisi de la maîtrise de l'action. On peut imaginer une situation intermédiaire, où la machine est un auxiliaire pédagogique, sollicité au gré du maître dans les limites fixées par lui. Cela se produit si l'activité avec la machine ne reçoit de signification que celle qu'apporte en dehors d'elle l'échange entre adulte et enfants. La machine montre, suggère, répète, réalise pourvu qu'on l'interroge au bon moment.

b) le statut de la réponse

Ce que l'on place le plus souvent sous la rubrique EAO relève de la fonction de répétiteur. Les réponses fournies par la machine sont des *évaluations* qui ont pour effet d'une part de sanctionner un comportement (une réponse de l'élève) et d'autre part d'orienter la suite de l'activité (questions suivantes). Cette évaluation-sanction (rédigée par avance dans le programme) réfère donc à une organisation pré-existante, et à un savoir constitué. C'est la conformité aux objets de ce champ de connaissance qui est énoncée ; l'évaluation est normative. A l'opposé, dans la programmation, les seules sanctions explicites du langage sont celles qui touchent aux règles du jeu (orthographe, syntaxe . . .). Pour le reste l'élève est lui-même juge de la conformité du résultat à son projet. C'est pourquoi on a parlé de la vertu constructive de l'erreur. *Si* l'élève se souvient de son projet, *s'il* n'en change pas et *s'il* reconnaît l'inadéquation, il en cherche les causes et ré-ajuste l'action.

La construction du savoir n'est probablement pas une condition suffisante d'apprentissage durablement établi. Une figure intermédiaire pourrait être celle-ci : l'évaluation n'est pas extérieure à l'enfant mais gouvernée par lui. Le projet par contre est intégré à un plan dont il n'a pas toute l'initiative. Ainsi conserve-t-il un "jeu" (constructif) sans qu'il en ignore ou perde le cap.

c) la modification des représentations

L'opposition essentielle provient d'une hypothèse sur la prise en compte des représentations. Dans le premier cas, on vise une modification des comportements associés (sans hypothèse supplémentaire sur ce qui les fonde) ; il faut que les opérations soient justes, les dictées sans faute, etc. Dans l'autre cas, on tient que la connaissance s'établit par ré-organisation (assimilation/accommodation) conséquente de l'échange élève/milieu. Mais dans ce cas, pourquoi *évacuer l'objet ?* un langage de programmation n'a pas pour objet explicite un domaine de connaissance (comme le calcul, l'histoire ou l'astronomie). Ce que l'enfant s'approprie ainsi, ce ne sont pas les objets de son choix (maison, fleur . . .) qu'il connaît déjà, mais des *règles de jeu.* On verra plus loin lesquelles.

Il n'est ni nécessaire ni suffisant pour l'acquisition d'une connaissance qu'elle soit établie par une construction du sujet : nous disposons tous de savoirs acquis de façon normative. Mais l'appropriation est certainement facilitante. Pour être durable l'acquisition doit s'établir sur des représentations nouvelles, et quelquefois des formulations instituées. Chacun sait que des enfants de CM connaissant la division, retournent devant des situations de problème inhabituelles à des algorithmes plus "primitifs". La représentation que l'on se fait d'une notion est donc déterminante pour la réalité de l'acquisition. C'est ici que l'ordinateur peut intervenir : en fournissant et en faisant exercer (de façon dynamique) des représentations qu'une *parole (fugitive)* ou une *image (statique)* sont impropres à fournir à l'intuition. L'exercice (libre, mais limité) des règles d'un jeu conduit à se représenter la notion qui est le *germe* de ce jeu ; ainsi les figures classiques du Go sont des *images* de mouvements potentiels : ko, shisho, etc.

L'ordinateur devient un objet pour faire parler, et pour faire imaginer, bref pour enrichir les représentations. Laissez un novice et un expert observer un échiquier en cours de partie. Puis faites-leur reconstruire de mémoire. Le novice essaie de reconstituer, avec peine, ce qui est pour lui un disparate. Pour l'expert, l'échiquier est un réseau de relations qu'il reconnaît, comprend et retient aisément.

II – ASPECTS DE LA PROGRAMMATION

1) concepts de base

Il n'est pas sûr que l'initiation à la programmation vise toujours dans la pratique un objectif clairement identifiable. Les comptes-rendus de telles entreprises se répandent habituellement en propos attendris sur l'enthousiasme inaltérable des enfants, la qualité de leurs échanges, la variété de leurs créations. On est plus volontiers tenté de justifier après coup ce qui se passe bien, que de trouver une nécessité à des travaux rébarbatifs. On s'accorde ainsi à trouver que Logo "structure l'espace", développe la rigueur et la persévérance. C'est clairement prendre la question à rebours. Si l'espace de l'élève n'est pas bien organisé, ses programmes (en Logo graphique) échoueront ; s'il n'est pas rigoureux aussi ; et s'il n'est pas persévérant, il abandonnera. La motivation, la fascination initiale peuvent sans doute faire franchir des difficultés qui auraient rebuté ; et par conséquent affermir pour un temps une confiance incertaine. La découverte spontanée de la "géométrie de tortue" permet *d'exercer* des situations spatiales, non de les enrichir à moins qu'une conceptualisation ne soit entreprise par ailleurs.

Par contre un langage de programmation permet de découvrir et de fréquenter quelques concepts originaux (ou développés de façon originale par l'informatique) qui ont peu à voir avec le calcul et rien avec la géométrie. Nous allons en faire une revue sommaire pour mentionner les raisons possibles de leur importance (y compris **hors** de l'informatique) et quelquefois de leur difficulté.

A) La première idée concerne la distinction entre état et action. Une action transforme un état (instant T1) en un autre (instant T2). Le rôle du temps a été délibérement gommé des définitions et procédures mathématiques depuis un siècle (cf. les définitions de fonctions et de "transformations" géométriques). Cette évacuation, pour fondée qu'elle soit, ajoute des difficultés d'apprentissage maintes fois reconnues.

Retenons en tout cas qu'un programme d'ordinateur est exécuté de façon strictement **séquentielle** ; et qu'une méthode de programmation vise à assujettir la pensée (qui ne l'est pas toujours) aux contraintes de la machine et notamment à celle-ci. En conséquence, les langages et concepts les plus élaborés sont ceux qui permettent (ou simulent) une relative évasion de cette stricte séquentialité.

Une dernière conséquence est celle-ci : la programmation dépend strictement de la capacité *d'anticiper*, de composer *temporellement* des schèmes, d'envisager une composition *inverse.*

B) La seconde idée concerne les structures de données. Il n'est pas indifférent, selon le traitement que l'on envisage, d'organiser les données de telle ou telle façon. Ces structures ont une importance capitale en informatique, mais aussi au-dehors. On retiendra 3 exemples : les listes, les tableaux, les arbres.
Ce sont des structures de classement que l'on rencontre dès l'école maternelle. C'est précisément en abordant tôt ces schémas qu'on peut les charger de sens et les rendre opératoires. Un tableau est d'abord un classement à double entrée, puis à lignes et colonnes numérotées. C'est enfin une matrice à 2 ou à plusieurs dimensions. A un niveau suffisant d'abstraction, on peut montrer l'équivalence (sous forme de graphe par exemple) de ces types de structure (arbre = liste de listes, etc. . .). Ces structures peuvent répondre aussi à une "réalité informatique". Ainsi la mémoire d'un ordinateur est-elle organisée comme une liste, et certains traitements requièrent-ils des listes particulières (piles ou files). Il n'est pas sans intérêt d'observer que la mémoire humaine à court terme fonctionne comme un tampon, et qu'elle est essentiellement verbale (organisée en liste) : gérée comme une pile ou comme une file ? sans doute n'est-ce pas aussi simple. Le mémoire à long terme, en tout cas, a peu à voir avec ces structures-là.

C) La troisième idée concerne les *fonctions et prédicats.* Les fonctions sont des procédures particulières en ce qu'elle ne mettent pas en jeu l'exécution d'une action, mais qu'elles "rendent" une valeur. On en sait l'importance en mathématique, et une programmation évoluée y recourt volontiers de façon à "mettre de côté" certains traitements particuliers. Les prédicats sont des cas particuliers qui "rendent" vrai ou faux. Ils interviennent notamment dans toutes les conditionnelles. Il semble probable en tout cas que la possibilité de recourir à des prédicats au cours d'une activité de programmation soit fortement liée à la capacité de description *verbale* de la situation.

D) La quatrième idée, certainement la plus importante,est celle de **procédure.** Elle mérite une station prolongée. Une procédure est un "morceau" de programme identifiable par son effet. C'est donc un concept qui permet de *classer* et de *hiérarchiser. Mais il est en rappo*rt avec la capacité de *découper,* c'est-à-dire de repérer des **motifs.** On retrouve ce concept à des niveaux très divers, depuis les actes élémentaires de la perception jusqu'à la résolution de problème (et la structure des ordinateurs). C'est dire que s'il intervient en informatique ce n'est pas tant *à cause* des contraintes de la machine, mais *en dépit d'elles* : c'est l'un des éléments de confort de la programmation évoluée.

Nous prendrons quelques exemples graphiques pour illustrer cette idée (sans en réduire la portée).

fig. 1

fig. 2

fig. 3

Les figures ci-dessus représentent des mouvements continus. On n'y repère aucune régularité d'ensemble. La figure 2 est formée de segments, la figure 3 aussi. A mieux regarder, on aperçoit un motif qui revient souvent. On peut ainsi la fractionner de façon à distinguer chaque occurence de ce motif. Reconnaître un objet est un acte perceptif de même nature. Une procédure consiste à distinguer des **unités de sens** (des "motifs"). Même chose en ce qui concerne l'action, ou la pensée : faire intervenir des unités de sens macroscopiques permet de traiter des images, des mouvements, ou un problème au **niveau** de signification qui semble optimal pour l'action ou la pensée. (cf. la théorie de la forme, les holons de A. Koestler. . .)

Il n'est pas inutile de distinguer ici ce que l'on appellera procédure-ouverte et procédure-fermée. Les procédures-fermées, ce sont des schèmes, ou des principes issus de l'expérience, ou des significations qui fonctionnent à partir d'un moment *comme des a priori* (permanence de l'objet, signification des mots usuels, compétence linguistique, schéma corporel, reconnaissance des visages). Dès lors que la procédure s'est "refermée", sa genèse est (heureusement) oubliée ; toute remise en cause (accidentelle ou volontaire) est difficile et dangereuse.

D'autres procédures identifiables demeurent "ouvertes", c'est-à-dire qu'elles sont utilisées à un moment et dans un but, mais qu'elles peuvent être modifiées, fusionner ou disparaître. C'est là que se rencontre la réalité informatique. On a une image des procédures-fermées par les logiciels "cablés", mais l'ordinateur actuellement ignore les procédures "refermées", ou semi-ouvertes, même si le langage de programmation permet de les simuler. Cette simulation ne correspond à aucune modification de fonctionnement pour la machine. C'est typiquement évident avec Logo : une procédure ne peut se passer d'une déclaration de paramètre. Alors que la reconnaissance d'un carré est très antérieure à l'estimation de la mesure de ses côtés, un carré est une **forme**, ce dont le programme ne rend compte que superficiellement.

C'est pourquoi nous distinguons encore "procédure-bloc" et "procédure-forme". Une procédure-bloc est, strictement,un objet identifiable et singulier ; comme la maison ci-contre.

fig. 4

On peut voir dans cette maison trois"blocs" : la façade (rectangle), le toit (triangle) et la porte (carré).

Deux problèmes importants se posent que nous ne développerons pas :

– celui de *l'identification* (pourquoi un triangle, et pas un chevron ? pourquoi un rectangle et pas un pentagone ?)

– celui du *recollement* (où finit le rectangle, où commence le triangle **donc** où commence le rectangle ?)

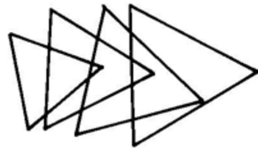


fig. 5

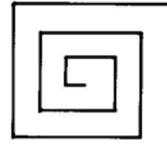


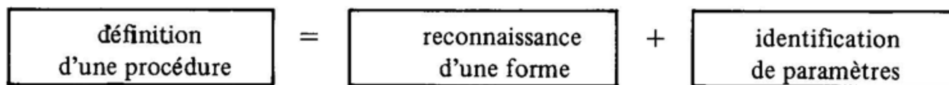
fig. 6

La figure 5 montre clairement qu'il s'agit de triangles équilatéraux (il y en a bien d'autres !). Ils sont tous différents (en taille et en orientation). De la même façon, la spirale (fig. 6) peut se traduire :

AV1 TD90 AV1 TD90 AV2 TD90 AV2 TD90 AV3 TD90. . .

où l'on repère des séquences (AV TD); soit S cette séquence ; et une progression : S1 S1 S2 S2 S3. . . ou même, si R_1 désigne :

Repete 2 (AV1 TD90), la progression R_1 R_2 R_3 . . .



En ce qui concerne la suite récursive des "flocons" de Von Koch, le repérage de la bonne procédure consiste à dire : on passe d'un état au suivant en ajoutant une pointe sur chaque segment (on ne mentionne pas la longueur du segment).

Bien entendu, ce repérage de forme est facile sur des supports graphiques. Il l'est beaucoup moins dans d'autres situations. Qu'on se rappelle la difficulté des enfants de C.P. avec la numération, lorsqu'il s'agit de groupements d'ordre N (N>2) : le "groupement d'ordre N" est *beaucoup* plus difficile à identifier que le groupement d'ordre 1. Ce cap franchi, la récurrence est comprise.

2) résolution de problème

Pourquoi s'intéresser ici à la résolution de problème ? D'abord parce qu'un apprentissage ne peut être validé que par un ré-investissement : une connaissance qui demeurerait indisponible et inutilisable manquerait de signification. En second lieu, on considère souvent que l'activité de programmation a quelque rapport avec la résolution de problème ; ce rapprochement mérite un examen. Enfin, l'informatique, à travers la discipline particulière qu'est l'intelligence artificielle éclaire la résolution de problème d'un jour nouveau.

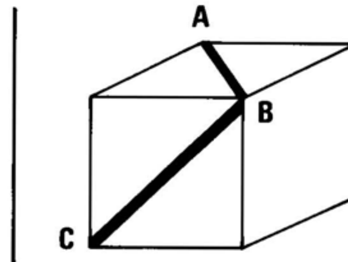
On entend ici par problème des situations très générales où il est nécessaire de mobiliser des connaissances pour apporter une réponse a priori non évidente à une question. Un problème peut être caractérisé par trois termes : une situation initiale (S_I), une situation finale (S_F) et entre elles un ensemble de transformations (matérielles ou symboliques). Résoudre un problème consiste à trouver un "chemin" dans cet espace entre S_I et S_F ; ce chemin peut résulter de tâtonnements opiniâtres, d'une intuition soudaine, ou d'une stratégie rationnelle. Mais force est de constater que la résolution de problème ne fait jamais l'objet d'un enseignement, et que l'exposé d'une solution, si elle gomme les tâtonnements et la justification des choix est d'un faible secours pour résoudre de nouveaux problèmes. G. Polya (How to solve it, 1945) déplorait cet état de choses. La question de savoir comment on résoud un problème, ce qu'il faut entendre par intuition, induction, analogie, et s'il peut en exister un apprentissage méthodique, est en débat depuis longtemps.

La question se pose en termes analogues pour la programmation. On sait le gaspillage formidable qu'occasionne une programmation mal structurée : programmes impossibles à relire et à adapter, détection impraticable des fautes, etc. La recherche d'une mémoire est essentielle. La construction d'un programme se présente comme un problème : partant d'un *projet* (l'idée que l'on se fait de l'action à entreprendre), il s'agit d'aboutir à un *programme* c'est à dire une description telle que la machine puisse l'exécuter. Les règles du jeu sont le code du langage : comment puis-je *transcrire* mon projet conformément à ces règles ? Il s'agit essentiellement d'un problème de *formulation* : on peut imaginer des états intermédiaires S_1. S_2... sémantiquement équivalents et prenant en compte progressivement les règles du jeu. C'est ce que préconise la méthode d' "analyse descendante". On divise la difficulté par un fractionnement réitéré des étapes, et à condition de clairement définir l'état d'entrée et de sortie des paramètres de chaque module.

Malheureusement ce type de problème, et les démarches qui lui sont adaptées, a peu à voir avec des situations comme celles-ci (à être si connues, sont-elles encore de problèmes ?) :

• • •
• • •
• • •

Il s'agit de trouver une ligne brisée de 4 segments qui passe une seule fois et une seule par ces neuf points.



Quelle est la mesure de l'angle $\widehat{ABC}$?

La résolution tient en peu d'opérations, voire une seule : la difficulté est de trouver une *direction* pour le premier pas. C'est ici que l'on parle *d'heuristique.* L'heuristique parait être du domaine du savoir-faire, de l'intuition, ce qui n'éclaire pas vraiment la question ... l'heuristique ne se laissant guère définir ou étudier. Deux attitudes consistent, l'une à tenter de la réduire, en fin de compte à un processus algorithmique, l'autre à examiner les points d'appui qu'elle offre malgré tout.

Le premier point de vue a été soutenu par le behaviorisme, et après lui par l'intelligence artificielle. Il est cohérent avec l'idée que l'on pourrait établir avec l'ordinateur une simulation satisfaisante de l'humaine résolution de problème. Ces études n'ont pas tout à fait répondu à leurs ambitions, même si elles ont donné, dans des domaines particuliers, des résultats appréciables (cryptarithmétique, jeu d'échec ...). Il semble en tout cas qu'on ne peut éviter de conclure à l'existence de stratégies *locales* et à l'inexistence d'une méthode *générale de résolution.*