DEUX TYPES D'ACTIVITÉS DE PROBATION SUR LE THÈME DE L'ÉCLUSE

Michel DUPONT

1. PRÉSENTATION

Les logiciels diffusés sous le titre d'ÉCLUSE peuvent être considérés de deux points de vue: résultat d'une activité de programmation proposée aux élèves ou point de départ d'une nouvelle activité qui doit aboutir à une simulation sur écran graphique du fonctionnement d'une écluse schématisée. Au cours de notre exposé, nous serons amenés à privilégier ce deuxième point de vue en proposant d'abord un logiciel de base que nous avons conçu à partir d'une critique des versions existantes puis des activités de programmation qui utilisent ce logiciel.

Ces activités s'adressent tout particulièrement aux élèves du Cours Moyen tant par les aptitudes intellectuelles que par les connaissances qu'elles requièrent ou qu'elles contribuent à mettre en place; l'objectif étant de "faire programmer les élèves dans une perspective logistique" c'est à dire de leur donner "une juste idée de la notion de **calcul** organisé et automatisable effectué sur un ensemble de données elles-mêmes organisées et produisant les effets voulus". Nous nous référons ici aux Compléments aux Programmes et Instructions de l'école élémentaire, texte qui évoque longuement le thème de l'écluse et que nous aurons plusieurs fois l'occasion de citer.

On peut d'ailleurs s'étonner qu'en dépit de ces encouragements officiels ce thème soit en fait bien peu utilisé par les enseignants. Il est vrai que la prise de contact avec ÉCLUSE s'est souvent faite dans de mauvaises conditions. Les enseignants, supposant qu'il s'agissait d'un moyen pour expliquer le fonctionnement d'une écluse, ont été bien déçus quand ils ont découvert cette péniche fantaisiste qui passe à travers les portes et se transforme en sous-marin ou en avion.

Or il faut bien reconnaître qu'ils n'ont pas toujours trouvé les explications nécessaires lors de ces stages du plan I.P.T. animés par des

enseignants eux-mêmes mal informés et ne disposant que de versions très critiquables.

2. CRITIQUE DES VERSIONS DIFFUSÉES DANS LES VALISES DU PLAN I.P.T.

Nous avons relevé entre autres

- des anomalies graphiques du genre "trous dans l'eau"
- une documentation qui ne correspond pas toujours au logiciel (l'exemple d'éclusage de la version Été 85 correspondait au logiciel de Pâques 85)
- des noms de macro-primitives qui sont parfois mal choisis (REMPLIR [BASSIN] qui laisse supposer qu'on pourrait remplir ou vider les autres parties en eau)
- la représentation de tous les volumes en eau avec une largeur identique qui engendre la même confusion. Les élèves veulent faire monter ou descendre l'eau dans chacun des trois volumes comme s'il s'agissait d'une succession d'écluses.
- la place consacrée au texte est insuffisante. Il est impossible d'écrire ou de visualiser une procédure en entier sans supprimer le graphisme.

3. FAUT-IL AVOIR RECOURS À UN PROGRAMME GÉNÉRATEUR D'OBJETS GRAPHIQUES ?

Par ailleurs le parti pris d'utiliser un programme générateur d'objets graphiques comme GENS est discutable. Cette démarche ne se justifie que lorsqu'on veut faire construire un tel logiciel par les élèves en partant d'un lot de procédures d'assistance au dessin qui seraient davantage à leur portée que les primitives du langage. Ce lot de procédure aurait alors valeur de langage intermédiaire entre le langage évolué utilisé et le sur-langage que les élèves auraient à construire.

Cette idée est séduisante, mais il convient de se demander si ce travail est à la portée des élèves. Lorsque nous avons voulu travailler dans ce sens, nous nous sommes en effet heurtés d'emblée à deux difficultés. D'une part la particularité de la carte graphique Thomson amène des effets de parasitage et de "trous dans l'eau" difficiles à maîtriser pour des élèves de ce niveau puisqu'il faut tenir compte du fait

que sur l'aie horizontal les points sont regroupés en octets ne pouvant contenir qu'une couleur de fond et/ou une couleur de forme. D'autre part, la manipulation des procédures de GENS suppose acquises un minimum de connaissances en géométrie euclidienne. Or, sur ce point, les avis divergent. Les auteurs de "144 procédures utiles en LOGO"(Ed. CEDIC, 1986) vont jusqu'à affirmer que ce niveau de difficulté ne doit être abordé qu'au Collège . Pour notre part, considérant que les programmes prévoient dès le cours élémentaire l'initiation au "repérage des cases et des nœuds d'un quadrillage" et "l'utilisation de ces repérages", nous ne sommes pas aussi catégoriques. II nous parait cependant prudent de limiter les pré-requis en mathématiques nécessaires à l'activité.

4. UN CHOIX QUI DÉCOULE D'UNE ORGANISATION RATIONNELLE DE LA PROGRESSION DES APPRENTISSAGES.

De plus, la programmation de l'ensemble du logiciel ne peut être proposée qu'à des élèves qui ont déjà des connaissances en ce qui concerne les grandes structures de base de la programmation et les spécificités du langage utilisé que ce soit un langage évolué traditionnel ou un "langage intermédiaire" comme GENE.

Or, précisément, l'un de nos objectifs est de permettre la découverte de ces structures de base sans avoir à passer par l'apprentissage exhaustif et formel d'un langage. Nous allons démontrer à l'aide d'exemples que cela est possible avec les deux types d'activités que nous proposons à partir de notre version de base. Notre souci d'établir une progression rationnelle des apprentissages nous conduit donc à reporter au moment où les élèves auront acquis les connaissances nécessaires ce troisième type d'activités : la construction et la réalisation de l'ensemble d'un projet, sur le thème de l'écluse ou sur un thème similaire, en utilisant, éventuellement des procédures d'assistance au dessin.

Il va de soi que ceux qui ont acquis par d'autres voies les connaissances et savoirs-faire nécessaires pourront commencer par ce troisième type d'activités et qu'ils trouveront dans les deux premiers l'occasion de mettre en œuvre leur compétence sur un sujet intéressant.

5. NOTRE VERSION DE BASE

On ne s'étonnera pas que nous ayons choisi LOGO pour écrire notre version de base car ce langage, entre autres avantages, se prête particulièrement bien par son caractère procédural à la création de sur-langages. Si nous suivons ainsi la voie tracée par les autres auteurs, il est évident par contre que nous n'avons pas recours comme eux à un programme générateur d'objets graphiques puisque cette version ne doit pas être considérée comme un exemple de production d'élève mais comme un outil qui doit leur être fourni. Elle a ainsi l'avantage supplémentaire d'être plus courte, ce qui n'est pas négligeable surtout pour ceux qui chargent leurs programmes avec un lecteur de cassette. Elle est aussi plus lisible car sa structure est des plus simple.

On trouve d'abord deux procédures outils qui sont rentables dans un logiciel qui présente un graphisme avec de nombreux angles droits : DAD (tourne à droite de 90°, avance de ..., tourne à droite de 90°) et GAG (id*, à gauche). Puis deux autres procédures-outils : PLA :P :Q et RECT :X :Y :C servent respectivement à placer la tortue sur un point de coordonnées :P et :Q et à dessiner un rectangle de dimension :X et :Y et de couleur :C

Enfin les six autres procédures ont valeur de macro-primitives

DEBUT :N pour la mise en place du paysage (schéma d'une écluse vue en coupe avec un bateau dont la position est déterminée par le paramètre :N qui doit être un nombre entier négatif pour que le bateau soit en amont de l'écluse, positif pour qu'il soit en aval et nul pour qu'il soit dans le bassin. S'il est plus petit que -2 ou plus grand que +2 le bateau sera en dehors de l'écran

- VERS-AVAL et VERS-AMONT pour déplacer le bateau
- VIDER et REMPLIR ; si le bateau est dans le bassin, ces macro-primitives entraînent également la montée et la descente du bateau.
- C commande l'ouverture et la fermeture des portes et des vannes. Cette macro-primitive doit être suivie d'une liste de trois mots. Exemple
- C [OUVRIR PORTE AMONT] ou C [OUVRIR VANNE AVAL]

On pourra par ailleurs remarquer que la méthode utilisée pour simuler les déplacements est classique (elle est expliquée dans divers

manuels), que le nombre des primitives employées est restreint et qu'aucune procédure n'est récursive. C'est dire qu'avec une initiation minime à LOGO tout enseignant pourra modifier cette version ou construire sur ce modèle d'autres programmes de simulation d'objets techniques sur écran graphique.

6. STATUT DES MACRO-PRIMITIVES DANS LES DEUX TYPES D'ACTIVITÉS

Les macro-primitives que nous avons créées ont un statut différent suivant l'un ou l'autre des deux types d'activités possibles :

- La programmation de la simulation d'un parcours du bateau qui aura été prévu à l'avance,
- La réalisation d'un programme de simulation dans lequel certaines manœuvres de l'éclusage se feront automatiquement et/ou qui sera plus ou moins protégé contre les erreurs de l'utilisateur, celui-ci gardant le libre choix du parcours.

Dans le premier cas, tout se passe pour l'élève comme s'il était en présence d'un langage qui aurait pour primitives l'ensemble constitué de nos six macro-primitives, des deux primitives POUR et FIN, et éventuellement de **q**uelques autres comme RAMENE, SAUVE, REPETE... C'est ce que nous avons appelé un sur-langage. De même qu'on ne modifie jamais les primitives d'un langage l'élève n'aura jamais à modifier les macro-primitives.

A l'inverse, dans le deuxième type d'activité, l'élève pourra éventuellement ajouter des lignes LOGO à l'intérieur des procédures qui génèrent les macro-primitives. Dans ce cas, en effet, notre version de base ne doit plus être considérée comme un sur-langage mais comme un programme inachevé que l'élève devra compléter pour réaliser son projet. Bien qu'elles aient alors un statut différent de celui des primitives nous pouvons continuer à dénommer macro-primitives ces mêmes procédures en accordant à ce terme le sens donné par le Complément aux Programmes et Instructions du 15 mai 85 qui précise que

"Si une procédure est d'utilité suffisamment générale pour être considérée à l'instar des primitives comme un outil que l'on a de grandes chances de réinvestir dans l'écriture d'autres procédures, on est justifié de lui donner le nom de macro-primitive."

7. PRÉ-REQUIS, CONNAISSANCES À CONSOLIDER, ACTIVITÉS PRÉALABLES OU COMPLÉMENTAIRES

Avant de commencer toute activité de programmation sur ce thème, les élèves devront connaître les principes du fonctionnement de l'écluse ce qui suppose des connaissances à la fois en géographie (écluse resituée dans son contexte) et en physique (principe des vases communicants). Rappelons, si cela est encore nécessaire, que les activités proposées en informatique sur le thème de l'écluse ne sont pas un moyen d'acquérir ces connaissances mais, qu'au contraire, celles-ci doivent être préalablement assimilées. Citons encore à ce sujet les Compléments aux Programmes et Instructions du 15 mai 85 qui indiquent que, dans le cas contraire, le pédagogue devra "recourir 'à toutes les activités relevant de la technologie au cours moyen (visite, recherche documentaire, expérimentations, productions de documents de synthèse parmi lesquels des schémas, voire des maquettes). Certaines de ces activités pourront même éloigner pendant quelques temps les élèves de leur projet de simulation". Et le même texte ajoute : "C'est pour cette raison que le thème choisi doit impérativement avoir un rapport clair avec les programmes du Cours Moyen". Remarquons que dans cette perspective le thème de l'écluse est particulièrement bien adapté.

Il arrivera sans doute au cours des diverses activités que l'enseignant découvre dans des productions d'élèves des bassins d'écluse qui se vident ou se remplissent alors que les vannes sont fermées ou d'autres anomalies. Ce sera alors l'occasion de faire quelques retours en arrière pour consolider des connaissances qui se seront avérées trop fragiles. L'élève devra également acquérir une certaine maîtrise du clavier, manipuler l'éditeur LOGO, connaître quelques primitives de LOGO parmi lesquelles POUR FIN REPETE ET SI EGAL? DONNE RENDS VRAI FAUX ainsi que la syntaxe qui s'y rapporte. Ces connaissances et savoirs-faire pourront être acquis au cours des activités proposées mais ils pourront aussi faire l'objet d'activités préalables ou complémentaires.

Par ailleurs, si le thème de l'écluse, déjà très expérimenté mérite d'être privilégié, il n'exclut pas le recours à d'autres thèmes pour faire programmer les élèves dans une perspective logistique. Bien que nous ayons sur ces différents points des propositions précises à faire, nous n'en dirons pas plus ici afin de ne pas sortir du cadre de notre exposé.

8. LE PREMIER TYPE D'ACTIVITÉ

On commence par présenter les macro-primitives aux élèves qui en découvrent les effets en mode immédiat. Il est alors permis d'obtenir des images incompatibles avec le phénomène réel. C'est le moment de transformer le bateau en avion ou en sous-marin ! On exploite ainsi dès ce premier contact la possibilité de validation graphique, avantage essentiel des logiciels de simulation d'objets techniques. A chaque étape de 'son travail, l'élève peut en effet obtenir l'image de ce qu'il a réellement construit; ce qui lui permet de confronter le résultat obtenu au projet qu'il avait formé.

Dans un deuxième temps, l'élève cherche à exécuter un éclusage correct. Il doit d'abord décrire l'algorithme séquentiel, c'est à dire la suite des **commandes** nécessaires à l'exécution de la simulation puis traduire cet algorithme en langage de programmation. Ce travail peut d'ailleurs être groupé en une seule étape tellement l'écriture avec notre sur-langage est proche de la description de l'algorithme. C'est pour obtenir cette similitude des deux écritures que nous avons choisi des noms de macro-primitives parfois longs mais toujours explicites.

Après une exécution "pas à pas", en mode immédiat, d'un éclusage correct on présente la possibilité de créer, avec les primitives POUR et FIN, une procédure. C'est à dire, dans ce cas, le regroupement sous un nom choisi par l'élève d'une séquence de commandes qui pourra être exécutée d'un seul bloc par le simple appel de son nom. Il est nécessaire à ce niveau que l'élève soit initié à l'utilisation de l'éditeur afin d'introduire facilement d'éventuelles corrections ou améliorations dans ses procédures.

L'élève peut ensuite créer plusieurs procédures comme MONTVAL (éclusage amont-aval) et VALMONT (éclusage aval-amont) puis les assembler dans une seule procédure qu'on pourrait appeler VA-VIENS

```
POUR VA-VIENS
DEBUT -1 MONTVAL VALMONT
FIN
```

(les procédures MONTVAL et VALMONT sont reproduites dans l'exemple suivant).

On obtient ainsi un premier programme hiérarchisé. Cette dernière procédure joue en effet un rôle de matrice puisqu'elle appelle,

après la macro-primitive DEBUT, deux procédures qui ne sont elles-mêmes constituées que de macro-primitives, ce qui les situe à un niveau inférieur dans la structure du programme.

Pour la dernière phase du travail, on demande à chaque élève de prévoir un déplacement plus important du bateau. Celui-ci est ensuite divisé en plusieurs étapes qui sont elles-mêmes encore divisées jusqu'à ce que chaque unité puisse s'écrire exclusivement avec les macro-primitives (et éventuellement la primitive REPETE). Ayant au fil de cette démarche décrit l'algorithme, l'élève peut ensuite écrire son programme avec notre sur-langage.

Cette dernière phase de travail permet non seulement l'évaluation des connaissances acquises précédemment mais elle en permet également le dépassement car la démarche employée ici est, en général, plus féconde que la précédente qui consistait à construire des blocs élémentaires à partir des macro-primitives pour construire ensuite, par juxtaposition, des blocs plus importants. Nous sommes partis de l'énoncé du problème pour aboutir à sa solution en utilisant l'"analyse descendante". Bien qu'il soit parfois nécessaire, surtout lors d'un apprentissage, de procéder du simple au complexe, comme nous l'avons fait dans un premier temps, il convient de privilégier cette autre démarche ainsi décrite dans le manuel qui accompagne les valises du plan I.P.T. (CNDP,1985) : "La démarche que l'on s'est accoutumé d'appeler "descendante", issue (bien après Descartes !) des études méthodologiques propres à l'informatique est non seulement fructueuse pour une programmation "saine", mais, bien au-delà, c'est une méthode féconde pour la résolution de problèmes. Elle consiste à diviser le problème (et à identifier ses parties) dès avant de savoir le traiter; puis à fractionner encore jusqu'à ce que l'énoncé (en langue naturelle, ou à l'aide d'un schéma) de chaque fragment ne pose plus à la traduction en langage de programmation de difficulté de structure".

Dans l'exemple reproduit ci-contre, nous avons introduit la primitive REPETE dans la procédure-matrice BALADE. On peut, en effet, à n'importe quelle étape de ce premier type d'activités, présenter, avec cette primitive, la notion d'itération.

```
POUR BALADE
DEBUT -1 MONTVAL SORTIE-D VALMONT SORTIE-G MONTVAL
REPETE 3 [VERS-AVAL]
FIN
```

```
POUR MONTVAL
C[OUVRIR PORTE AMONT] VERS-AVAL C[FERMER PORTE
AMONT]
C[OUVRIR VANNE AVAL] VIDER C[OUVRIR PORTE AVAL] VERS-
AVAL
C[FERMER PORTE AVAL]
FIN

POUR VALMONT
C[OUVRIR PORTE AVAL] VERS-AMONT C[FERMER PORTE AVAL]
C[OUVRIR VANNE AMONT] REMPLIR C[OUVRIR PORTE AMONT]
VERS-AMONT C[ FERMER PORTE AMONT]
FIN

POUR SORTIE-D
VERS-AVAL VERS-AVAL VERS-AMONT VERS-AMONT
FIN

POUR SORTIE-G
VERS-AMONT VERS-AMONT VERS-AVAL VERS-AVAL
FIN
```

On peut également présenter la récursivité. Toutefois, cette propriété qui n'est propre qu'à quelques langages de programmation nous paraît difficile à comprendre pour des élèves de cours moyen. Nous la réservons à des élèves plus âgés (il pourra s'agit, par exemple, d'enseignants en stage).

Après avoir constaté que la procédure SANS-FIN boucle sur elle-même, on constatera l'équivalence de la procédure itérative ALLER-RETOUR1 et de la procédure ALLER-RETOUR2 qui contient un appel récursif avec un compteur.

```
POUR SANS-FIN
MONTVAL VALMONT SANS-FIN
FIN

POUR ALLER-RETOUR1 :N
REPETE :N [MONTVAL VALMONT]
FIN
```

```
POUR ALLER-RETOUR2 :N
SI EGAL? :N 0 [STOP] [MONTVAL VALMONT ALLER-RETOUR2 :N -
1]
FIN
```

9. LE DEUXIÈME TYPE D'ACTIVITÉS

Avec ce deuxième type d'activités, nous proposons à l'élève de compléter la version de base en ajoutant des lignes LOGO dans les procédures qui génèrent les macro-primitives ou en écrivant de nouvelles procédures.

L'élève partira d'un projet qui pourra contenir des extensions de deux natures

- - protection contre les éventuelles erreurs de l'utilisateur,
- - automatisation de certaines opérations de l'éclusage.

9.1.: créer des tests et des messages d'erreur

Envisageons d'abord le cas où l'élève veut créer ce que nous appelons des protections, c'est à dire une analyse du message de l'exécutant qui amène éventuellement l'ordinateur à retourner un message d'erreur. C'est l'occasion de découvrir la notion de test qui est fondamentale en programmation. En plus des primitives qui commandent l'affichage du message d'erreur (EC FCT.J, deux nouvelles primitives sont nécessaires

- SI avec ses deux syntaxes :

SI <prédicat> [ceci], SI <prédicat> [ceci][sinon cela] La seconde sera utilisée au paragraphe 9.4 pour créer des prédicats.

- ÉGAL? qui autorise également deux syntaxes mais cela pour un résultat identique. Nous avons en effet le choix entre l'écriture préfixée (SI EGAL? <machin> <truc>) et l'écriture infixée (SI <machin> = <truc>). Dans la suite, nous nous en tiendrons à l'écriture préfixée, la seule qui soit utilisable avec les autres prédicats. Mais là aussi, les avis divergent...

Un premier niveau d'analyse, exclusivement applicable aux procédures paramétrées, consiste à vérifier que le(s) paramètre(s)

est(/sont) dans les normes attendues. En voici deux exemples qui, tous les deux, nécessitent quelques primitives supplémentaires

```
SI NON MEMBRE? :N [-3 -2 -1 0 1 2 3] [ERREUR STOP]
```

Cette ligne placée en tête de la macro-primitive DEBUT renvoie à la procédure ERREUR

```
POUR ERREUR
FCT 6 EC [ Après DEBUT il faut écrire l'un des nombres -3 -2 -1 0 1 2 3]
EC [Avec 3 ou -3, le bateau sera en dehors de l'écran.] FCT 3
FIN
```

La procédure ANALYSE (ci-dessous) est appelée depuis le début de la macro-primitive C

```
POUR ANALYSE :LIST
FCT 6 SI NON ÉGAL? COMPTE :LIST 3 [EC [Avec C il faut une liste de
    3 mots.] STOP]
SI ET NON EGAL? PREM :LIST "OUVRIR NON EGAL? PREM :LIST
    "FERMER [EC [Comme premier mot de la liste, vous ne pouvez écrire
    que OUVRIR ou FERMER.] STOP]
SI ET NON EGAL? ITEM 2 :LIST "PORTE NON EGAL? ITEM 2 :LIST
    "VANNE [EC [Comme deuxième mot de la liste, vous ne pouvez écrire
    que PORTE ou VANNE.] STOP]
SI ET NON EGAL? DER :LIST "AVAL NON ÉGAL? DER :LIST
    «AMONT [ EC [Comme troisième mot de la liste, vous ne pouvez
    écrire que AVAL ou AMONT] STOP ] FCT 3
FIN
```

Les élèves pourront être conduits à introduire cette seconde protection après avoir constaté qu'avec une erreur comme C[OUVRIR PORTE] on obtient l'ouverture de !a porte aval ou qu'avec C[OUVRIR PIRTE AMONT], c'est la vanne qui s'ouvre.

Un deuxième niveau d'analyse permet d'éviter des aberrations de situation en renvoyant des messages tels que :

"La porte (aval / amont) ne peut pas être ouverte quand le bassin est (plein / vide).", "Pour (remplir / vider) le bassin, la vanne (amont / aval) doit être ouverte". "Les deux vannes ne peuvent pas être ouvertes en même temps.", "Le bateau ne peut pas entrer dans le bassin puisque

la porte est fermée", "Avant d'ouvrir la porte, il faut (remplir / vider) le bassin. "

Il faudra alors créer des prédicats. (voir plus bas, 9.4)

9.2. Construire un projet

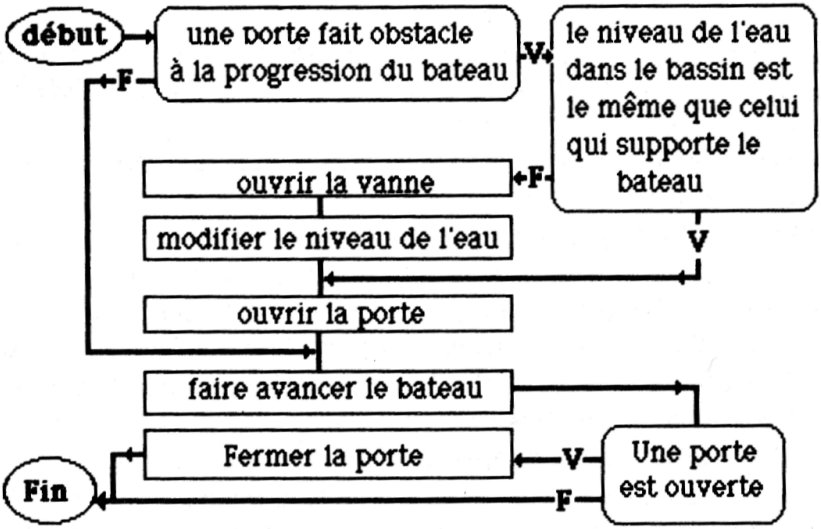
Nous allons en effet envisager maintenant le cas où l'élève est amené à compléter la version de base en automatisant certaines manœuvres de l'éclusage. Il est même possible de créer une version entièrement automatique. C'est avec ce projet que nous allons illustrer la méthode. Il s'agit de construire un programme qui se lancera avec INIT suivi d'un paramètre identique à celui de la macro-primitive DEBUT. Ensuite l'exécutant n'aura plus qu'à déplacer le bateau avec V-AVAL et V-AMONT. Le programme effectuera automatiquement les manœuvres d'ouverture et de fermeture des portes et des vannes ainsi que les modifications du niveau de l'eau dans le bassin.

9.3. Décrire l'algorithme

Une première description de l'algorithme peut être faite en quelques phrases :

- - si une porte fait obstacle au déplacement du bateau, il faut ouvrir cette porte.
- - Mais, si le niveau de l'eau dans le bassin n'est pas le même que celui du bief qui supporte le bateau, il faut d'abord ouvrir la vanne puis modifier le niveau de l'eau.
- - Le bateau effectue son déplacement.
- - Si une porte a été ouverte, il faut la refermer.

Cet algorithme contient trois niveaux de tests dont deux sont situés avant le déplacement et le troisième après. On peut faire apparaître cela avec l'organigramme ci-dessous. A chacun de ces trois niveaux correspondent plusieurs tests. En les écrivant tous, nous obtenons une nouvelle description de l'algorithme plus proche de l'écriture en langage de programmation.



1 ° niveau de test (avant le déplacement). * Le bateau avance vers l'aval

- s'il est à la position -1, il faut ouvrir la porte amont
- s'il est à la position 0, il faut ouvrir le porte aval

* Le bateau avance vers l'amont

- s'il est à la position +1, il faut ouvrir la porte aval
- s'il est à la position 0, il faut ouvrir la porte amont.

2° niveau de test (avant le déplacement) *l Ouverture de la porte aval

- Si le bassin est plein, il faut ouvrir la vanne aval puis vider le bassin

* Ouverture de la porte amont

- si le bassin est vide , il faut ouvrir la vanne amont puis remplir le bassin.

3° niveau de test (après le déplacement)

- si la porte aval est ouverte, il faut fermer la porte aval.
- si la porte amont est ouverte, il faut fermer la porte amont

9.4 Créer des prédicats

II va donc falloir tester la position du bateau, l'état du bassin (plein ou vide), l'état des portes (ouvertes ou fermées). Il faut pour cela créer

des prédicats. La technique à utiliser ne peut pas être devinée ni découverte par hasard. Si elle n'est pas connue des élèves, ce sera l'occasion de leur présenter de nouvelles primitives LOGO: RENDS, VRAI et FAUX.

Nous appellerons POSHORIZ? :Y (position horizontale) le premier prédicat dont nous avons besoin. Il permet de savoir si le bateau est à la position indiquée par le paramètre :Y

- Constater l'existence de l'objet (en BASIC, on dirait la variable) :HORIZ. En mode immédiat, quand on entre :HORIZ, l'ordinateur répond "QUE FAIRE DE .." en reprenant avec le codage utilisé dans la macro-primitive DEBUT le paramètre qui indique la position du bateau tel qu'il est représenté sur l'écran

- Entrer la procédure

POUR POSHORIZ? :Y

SI EGAL? :HORIZ :Y [RENDS VRAI][RENDS FAUX]

FIN

- Constater, toujours en mode immédiat, que l'ordinateur répond "QUE FAIRE DE VRAI" ou "QUE FAIRE DE FAUX" quand on l'interroge avec ce prédicat sous la forme , par exemple, POSHORIZ? -2 et que la réponse (VRAI ou FAUX) correspond bien à la position du bateau sur l'écran.

Nous avons maintenant tout ce qu'il faut pour écrire en LOGO les quatre premières lignes de notre algorithme. La première s'écrira

SI POSHORIZ? -1 [C[OUVRIR PORTE AMONT]]

9.5 Créer de nouveaux objets LOGO

Pour construire les autres prédicats, nous ne disposons pas des objets (ou variables) équivalents à :HORIZ. Il faut donc les créer. C'est l'occasion de découvrir la primitive DONNE qui permet l'affectation. En LOGO, on dit qu'on "donne une chose à, un objet". En BASIC, on dirait qu'on affecte une valeur à une variable. Nous allons ainsi créer l'objet :BASSIN qui aura pour chose "PLEIN ou "VIDE. Pour cela, il suffit d'introduire les deux lignes DONNE "BASSIN "PLEIN et DONNE "BASSIN "VIDE que l'on peut placer dans les macro-primitives REMPLIR et VIDER. On peut préférer construire deux nouvelles procédures *REMPLIR et *VIDER. Ce qui a l'avantage de ne faire manipuler par les élèves que des éléments qu'ils maîtrisent :

```
POUR *REMPLIR
REMPLIR DONNE "BASSIN "PLEIN
FIN

POUR *VIDER
VIDER DONNE "BASSIN "VIDE
FIN
```

Toujours avec la même méthode, on pourra vérifier en mode immédiat l'existence de ces nouveaux objets. Quand on entre BASSIN, l'ordinateur répond "QUE FAIRE DE VIDE °(ou de PLEIN). On peut ensuite construire soit un prédicat qui aura pour paramètre "PLEIN ou "VIDE, soit deux prédicats sans paramètres :

```
POUR BASSIN? :X
SI ÉGAL? :BASSIN :X [RENDS VRAI] [RENDS FAUX]
FIN

POUR BASSIN.VIDE?
SI ÉGAL? :BASSIN "VIDE [RENDS VRAI] [RENDS FAUX]
FIN

POUR BASSIMPLEIN?
SI ÉGAL? :BASSIN "PLEIN [RENDS VRAI] [RENDS FAUX]
FIN
```

9.6 Écrire le programme avec une initialisation

Avec la même méthode, on crée de nouveaux objets :PORTEAMONT et PORTEAVAL qui ont pour chose "OUV et "FER et de nouveaux prédicats OUVERTE.PORTE.AMONT? et OUVERTE.PORTE.AVAL? (ou un seul prédicat avec paramètre). On peut vérifier, en mode immédiat l'existence de ces nouveaux objets et de ces nouveaux prédicats puis il ne reste plus qu'à écrire en LOGO les 8 tests de notre algorithme en les plaçant dans des procédures appropriées. C'est presque terminé !

Si nous lançons le programme avec DÉBUT puis V-AVAL ou V-AMONT, l'ordinateur nous retourne le message

DANS (telle procédure) PAS DE CHOSE DONNES A (tel objet)

En effet, il faut toujours donner une valeur (chose) initiale à chaque variable (objet) utilisée dans le programme. D'où cette procédure

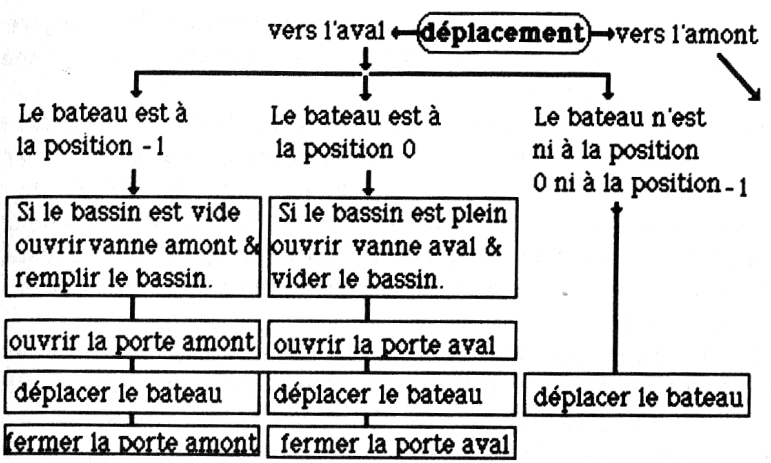
```
POUR INIT :N
DONNE "BASSIN "PLEIN
DONNE "PORTEAMONT "FER
DONNE "PORTEAVAL "FER
DÉBUT :N
FIN
```

Cette fois-ci, le programme est prêt (voir annexe 2).

9.7 Une version plus courte

Pour réaliser le même projet, d'autres algorithmes peuvent être élaborés. Nous en décrivons un autre ci-dessous avec un organigramme dont la forme ne respecte aucune règle conventionnelle.

Pour passer à l'écriture LOGO, il faut avoir recours à la primitive STOP et reprendre quelques procédures de l'exemple précédent (*REMPLIR *VIDER BASSIMPLEIN? BASSIN.VIDE? POSHORIZ? INIT). La procédure INIT sera simplifiée car seul l'objet BASSIN doit être initialisé. On obtient les deux procédures AVAL et AMONT ci-dessous. Le programme obtenu avec cette deuxième version est nettement plus court car il n'est pas nécessaire de tester l'état des portes. A l'inverse, d'autres algorithmes pourraient nous amener à tester non seulement l'état des portes mais aussi celui des vannes.



```
POUR AVAL
SI POSHORIZ? -1 [ SI BASSIN.VIDE? [ C[OUVRIR VANNE AMONT]
    *REMPLIR] C[OUVRIR PORTE AMONT] VERS-AVAL C[FERMER
    PORTE AMONT] STOP]
SI POSHORIZ? 0 [SI BASSIMPLEIN? [ C[OUVRIR VANNE AVAL]
    *VIDER]
C[OUVRIR PORTE AVAL] VERS-AVAL C[FERMER PORTE AVAL]
    STOP] VERS-AVAL
FIN

POUR AMONT
SI POSHORIZ? 1 [ SI BASSIMPLEIN? [ C[OUVRIR VANNE AVAL]
    *VIDER] COUVRIR PORTE AVAL] VERS-AMONT C[FERMER
    PORTE AVAL] STOP]
SI POSHORIZ? 0 [SI BASSIN.VIDE? [ C[OUVRIR VANNE AMONT]
    *REMPLIR]
C[OUVRIR PORTE AMONT] VERS-AMONT C[FERMER PORTE
AMONT] STOP] VERS-AMONT
FIN
```

10. CONCLUSIONS

Nous avons montré qu'en utilisant nos six macro-primitives et quelques primitives LOGO les élèves peuvent écrire des programmes hiérarchisés, construire des prédicats et des tests, effectuer des affectations, des itérations, des initialisations. C'est ce que nous appelons découvrir ces grandes structures de base qui sont fondamentales en programmation et qui, au-delà du langage utilisé, ont une valeur universelle.

Tout en illustrant notre propos par des exemples, nous avons mis l'accent sur la démarche. Construction et formulation du projet, analyse du problème, description de l'algorithme, dessin de schémas et d'organigrammes sont des étapes importantes qui précèdent l'écriture en langage de programmation, elle-même suivie par les essais, et éventuellement l'analyse et la correction des erreurs.

La possibilité de validation graphique facilite la phase des essais et le recours à des macro-primitives simplifie l'écriture en langage de programmation. Ainsi débarrassé de la plus grande partie des tâches les

plus fastidieuses et les moins formatrices que doit habituellement accomplir un programmeur, l'élève peut "programmer dans une perspective logistique", sans avoir à apprendre toutes les règles, la syntaxe et le vocabulaire d'un langage traditionnel.

Tel est l'essentiel de ce que nous avons voulu démontrer. Nous souhaitons en effet, qu'au delà des exemples donnés, le lecteur retienne surtout ces quelques conclusions. L'enseignant devra d'ailleurs non seulement définir lui-même les activités qu'il propose en fonction de la situation dans laquelle il intervient, du niveau et du rythme de ses élèves... Mais il lui appartient également de fournir une version de base qui soit adaptée. Il pourra éventuellement y inclure quelques-unes des extensions que nous avons présentées : protections, nouveaux objets LOGO, prédicats. Il devra cependant se garder d'approcher la fermeture complète avec une version trop automatisée et protégée; ce qui rendrait inutile l'ensemble de la démarche. Citons encore une fois à ce sujet les Compléments aux Programmes et Instructions du 15 Mai 85 :

"... il faut en particulier se méfier de toute détection d'erreur sophistiquée et de toute aide supplémentaire qui amènerait très vite l'élève à considérer qu'il se trouve devant une forme vicieuse de logiciel d'apprentissage du fonctionnement de l'objet, ou pire encore qui lui laisserait croire que l'ordinateur sait à l'avance quel est le bon algorithme."

Il convient par ailleurs de rappeler que nous n'avons pas traité exhaustivement le sujet. Nous avons en effet laissé de côté ce que nous avons appelé le troisième type d'activités; c'est à dire la construction et la réalisation de l'ensemble d'un projet en partant de procédures d'assistance au dessin et à l'animation graphique. C'est pourquoi, même si notre point de vue est critique à l'égard des versions diffusées dans les valises du plan I.P.T., notre travail ne les concurrence pas. Il constitue seulement une **contribution à un thème d'étude** qui est loin d'être épuisé. Nous espérons avoir ainsi fourni des outils et des idées qui seront utiles à nos collègues.

Michel DUPONT
AVRANCHES, 1987

ANNEXE 1 : VERSION DE BASE

```
POUR DAD :D
TD 90 AV :D TD 90
FIN

POUR GAG :G
TG 90 AV :G TG 90
FIN

POUR PLA :P :Q
LC FPOS PH :P :Q FCAP 0 BC
FIN

POUR RECT :X :Y :C
FCC :C
REPETE DIFF (QUOT :Y 2 ) 1 [AV ( :X - 1) GAG 1 AV ( :X - 1) DAD 1 ]
AV ( :X-1) GAG 1 AV (:X-1)
FIN

POUR DEBUT :N
DONNE "HORIZ :N SI PLG? :N 0 [DONNE "VERTI 13] [DONNE "VERTI 37] FEN CT ME 9 FCFG 6 FCB 7 FCFT H FCT 3 PLA -160 -27 TD 90 RECT 320 8 1 DAD 1 RECT 320 32 4 DAD 1 RECT 176 24 2 C[FERMER PORTE AVAL] C[FERMER PORTE AMONT] PLA SOMME -1 PROD :HORIZ 72 :VERTI RECT 8 16 5
FIN

POUR REMPLIR
PLA -32 13 TD 90 SI EGAL? MORE H [REPETE 24 [FCC 4 AV 47 LC GAG 8 AV 16 BC FCC 5 AV 15 LC AV 16 GAG 7 BC] DONNE "VERTI 37] [RECT 48 24 4]
FIN

POUR VIDER
PLA 15 36 TG 90 SI EGAL? MORE H [REPETE 24 [FCC -7 AV 16 FCC 5 AV
16 FCC -7 AV 15 DAD 8 AV 47 DAD 9] FCC 4 AV 1 DONNE "VERTI 13] [RECT 48 24 -7]
FIN
```

```
POUR VERS-AVAL
PLA SOMME -16 PROD :HORIZ 72 :VERTI REPETE 72 IFCC -7 AV 8
DAD 16 AV 1 FCC 5 AV 7 DAD 15] DONNE "HORIZ SOMME :HORIZ 1
FIN

POUR VERS-AMONT
PLA SOMME -1 PROD :HORIZ 72 :VERTI REPETE 72 [FCC -7 AV 8
GAG 16 AV 1 FCC 5 AV 7 GAG 15] DONNE "HORIZ DIFF :HORIZ 1
FIN

POUR C :LIST
SI EGAL? DER :LIST "AMONT [PLA -33 -19 DONNE "H 48] [PLA 23 –
19 DONNE "H 24] SI EGAL? PREM :LIST "OUVRIR [FCC4] [FCC3] AV
8 SI EGAL? ITEM 2 :LIST "PORTE [SI EGAL? PREM :LIST "FERMER
[RECT 64 8 1] [AV :H RECT (64 - :H) 8 –7] ] EFN "              /
FIN
```

ANNEXE 2 : VERSION AUTOMATIQUE

```
POUR INIT :N
DONNE "BASSIN "PLEIN
DONNE "PORTEAMONT "FER
DONNE "PORTEAVAL
"FER DEBUT :N
FIN

POUR V-AVAL
SI POSHORIZ? -1 [*C[OUVRIR PORTE AMONT ] 1
SI POSHORIZ? 0 [*C[OUVRIR PORTE AVAL] 1 VERS-AVAL
SI OUVERTE-PORTE.AMONT? [*C[FERMER PORTE AMONT ] ]
SI OUVERTE.PORTEAVAL? [*C[FERMER PORTE AVAL] 1
FIN

POUR V-AMONT
SI POSHORIZ? 1 [*C[OUVRIR PORTE AVAL ] ]
SI POSHORIZ? 0 [*C[OUVRIR PORTE AMONT ] 1 VERS-AMONT
SI OUVERTE.PORTE.AMONT? [*C[FERMER PORTE AMONT] 1
SI OUVERTE.PORTE.AVAL? [*C[FERMER PORTE AVAL] 1
FIN
```

```
POUR *REMPLIR
REMPLIR DONNE "BASSIN "PLEIN
FIN

POUR *VIDER
VIDER DONNE "BASSIN "VIDE
FIN

POUR *C :LIST
SI ET EGAL? :LIST [OUVRIR PORTE AMONT] BASSIN.VIDE? [C
   [OUVRIR VANNE AMONT] *REMPLIR ]
SI ET EGAL? :LIST [OUVRIR PORTE AVAL] BASSIMPLEIN ? [C
   [OUVRIR VANNE AVAL] *VIDER] C :LIST
SI EGAL :LIST [OUVRIR PORTE AMONT] [DONNE "PORTEAMONT
   "OUV]
SI EGAL :LIST [OUVRIR PORTE AVAL] [DONNE "PORTEAVAL
   "OUV]
SI ÉGAL :LIST [FERMER PORTE AMONT] [DONNE "PORTEAMONT
   "FER]
SI EGAL :LIST [FERMER PORTE AVAL] [DONNE "PORTEAVAL
   "FER]
FIN

POUR POSHORIZ? :Y
SI EGAL? :HORIZ :Y [RENDS VRAI] [RENDS FAUX]
FIN

POUR BASSIN.VIDE?
SI EGAL? :BASSIN "VIDE [RENDS VRAI] [RENDS FAUX]
FIN

POUR BASSIMPLEIN?
SI EGAL? :BASSIN "PLEIN [RENDS VRAI] [RENDS FAUX]
FIN

POUR OUVERTEPORTEAMONT?
SI EGAL? :PORTEAMONT "OUV [RENDS VRAI] [RENDS FAUX]
FIN

POUR OUVERTEPORTE.AVAL?
SI ÉGAL? :PORTEAVAL "OUV [RENDS VRAI] [RENDS FAUX]
FIN
```