MASTER EN SCIENCES INFORMATIQUES

E.A.O. Projet d'aide à l'étude et à la compréhension des fractions pour l'enseignement primaire

Verriest, Alain

Award date:
1985

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix (Namur)

Institut d'Informatique

E . A . O .

PROJET D'AIDE

A L'ETUDE ET A LA COMPREHENSION

DES FRACTIONS

POUR L'ENSEIGNEMENT PRIMAIRE.

Mémoire présenté par

Alain VERRIEST

en vue de l'obtention
du titre de
licencié et maître en informatique

Promoteur C. CHERTON

Année académique 1984 - 1985

Remerciements

Avant tout, je tiens à remercier pour sa patience, sa disponibilité et ses conseils, Monsieur Cherton qui a accepté la direction de ce mémoire.

Ma gratitude va ensuite à Monsieur Debrabandère, instituteur à Tournai, pour sa précieuse collaboration. Les membres du personnel pédagogique de l'institut Saint-Charles ont témoigné beaucoup d'intérêt à ce travail, je les en remercie également.

Merci enfin à Monsieur Y. Deville, aux parents et amis qui, de près ou de loin, ont contribué à l'élaboration de ce travail, et plus particulièrement à Mademoiselle B. Delsinne pour l'aide qu'elle m'a apportée.

TABLE DES MATIERES

INTRODUCTION

Ce mémoire traite d'un projet d'Enseignement Assisté par Ordinateur visant à améliorer l'étude et la compréhension de la notion de fraction dans le cadre de l'enseignement primaire, et plus particulièrement à partir du niveau de la quatrième année.

Beaucoup considèrent aujourd'hui que l'informatique constitue une branche d'avenir. Néanmoins, au cours de l'élaboration de ce travail, nous avons rencontré des enseignants et des parents pour qui l'utilisation de l'ordinateur ne représente qu'une mode passagère et qui ne comprennent pas l'intérêt que cet outil peut apporter au sein de l'école. D'autres, enfin, impressionnés par l'ésotérisme qui entoure l'informatique n'essaient même pas de se recycler dans ce domaine.

Le premier chapitre de ce mémoire permettra de situer notre démarche parmi l'éventail des possibilités existantes en E.A.O.. Les chapitres suivants seront consacrés à la présentation du projet au travers des différentes étapes de son cycle de vie : le chapitre 2 traitera de l'élaboration du projet et de l'analyse fonctionnelle; le chapitre 3 présentera la conception des modules les plus significatifs issus de cette analyse, et le chapitre 4 soulignera les détails d'implémentation du projet.

Que ce travail destiné à servir à la fois de mémoire de fin d'études et de projet en E.A.O., permette, modestement, de démystifier l'utilisation de l'ordinateur. Quant aux informaticiens, ils trouveront, dans ces pages, le détail de la réalisation d'un exemple de projet d'Enseignement Assisté par Ordinateur.

CHAPITRE 1

QU'EST-CE QUE L'ENSEIGNEMENT ASSISTE PAR ORDINATEUR ?

1. INTRODUCTION

L'introduction de l'informatique dans l'école poursuit différents buts :

- L'informatique peut être considérée comme un moyen de gestion dans l'établissement : ceci constitue une importante application de l'informatique et se caractérise par l'exécution de programmes de calculs qui permettent l'édition de factures, de bordereaux comptables, de fichiers d'élèves, etc.

- L'informatique peut également être utilisée en tant qu'outil pédagogique réservé à l'usage exclusif des enseignants, (constitution de bibliothèques, entraînement à de nouvelles techniques, etc.).

- L'informatique doit également pénétrer dans l'école comme matière d'enseignement, au même titre que d'autres matières scientifiques comme la physique, la biologie ou la chimie.

- Enfin, l'informatique peut aussi être utilisée comme moyen pédagogique à l'usage des élèves. On appelle cette pratique d'enseignement "Enseignement Assisté par Ordinateur" ou encore en abrégé E.A.O. .

C'est cette dernière forme d'introduction de l'informatique dans l'école qui nous intéressera dans le cadre de ce travail. Nous examinerons d'abord en quoi consiste l'E.A.O. et les divers types d'enseignement assisté par ordinateur. Nous étudierons ensuite les conditions préalables à sa bonne utilisation et enfin, nous observerons la situation actuelle belge ainsi que ce que nous propose l'Education Nationale.

2. LES DIVERS TYPES D'ENSEIGNEMENT ASSISTE PAR ORDINATEUR

Nous distinguerons d'abord l'utilisation de l'ordinateur par les élèves en tant qu'instrument d'enseignement individualisé, de l'utilisation de l'ordinateur par l'enseignant en tant qu'outil pédagogique nouveau dans un enseignement traditionnel.

2.1. Enseignement individualisé

Nous pensons qu'il faut à nouveau distinguer ici le cours proprement dit des activités individuelles des élèves destinées à fixer et à contrôler les connaissances acquises dans les cours.

1. Cours individualisé

Il s'agira ici d'une utilisation de l'ordinateur en vue de présenter et d'expliquer, sans la présence continuelle d'un professeur, de nouveaux concepts, de nouvelles connaissances ou de nouvelles techniques. Elle est également adaptée à un enseignement destiné à des personnes qui ne peuvent pas être présentes simultanément au même endroit.

2. Fixation et vérification des connaissances

Il s'agit ici de l'utilisation de l'ordinateur en vue de la manipulation et de l'assimilation profonde de concepts présentés antérieurement (d'une manière quelconque). Dans cette approche, l'ordinateur pourra être utilisé pour poser des questions-problèmes mettant en oeuvre la nouvelle notion supposée acquise.

Un didacticiel bien conçu permettra à l'élève d'explorer les divers aspects du concept enseigné; il sera à même de suivre son cheminement dans la recherche des solutions, et de réagir aux erreurs. Il choisira les problèmes posés en fonction du modèle qu'il se fait de l'élève sur base des réponses aux questions posées antérieurement. Le didacticiel ne pourra pas être un carcan obligeant l'élève à passer exactement par les étapes programmées mais lui laissant le plus d'initiative possible dans la recherche des solutions. L'enseignant reste toujours un recours dans le cas où le logiciel n'arrive plus à suivre le cheminement de l'élève.

On peut concevoir cette approche comme des travaux pratiques relatifs au cours. Les travaux pratiques exigent toujours une démarche constructive personnelle (ou par petit groupe). L'ordinateur qui met l'enseignant-garde-fou à la disposition permanente de l'élève nous semble pouvoir jouer ici un rôle plus important que dans le cours lui-même. Il permettra éventuellement aux élèves les plus avancés d'explorer des situations plus complexes.

Certains logiciels mettront particulièrement l'accent sur l'aspect entraînement ("drill and practice"), d'autres s'intéresseront à détecter les faiblesses des élèves et leur proposeront des remédiations, d'autres encore s'attacheront simplement à contrôler si des connaissances sont acquises soit pour permettre à l'élève de vérifier son niveau d'assimilation de la matière, soit pour permettre à l'enseignant de procéder à des interrogations automatiques.

2.2. L'ordinateur outil pédagogique de l'enseignant

L'ordinateur est ici utilisé par l'enseignant pour illustrer les concepts qu'il présente. Au même titre que d'autres outils (tableau noir, rétroprojecteur, enregistreur, vidéo, ...), l'ordinateur permet aux enseignants de présenter et d'illustrer les nouveaux concepts. L'apport principal de l'ordinateur permet de réagir rapidement à des données construites sur le moment : élaboration de schémas, graphiques, statistiques, etc... Le professeur peut visualiser directement l'effet des propositions formulées par les élèves. Un modèle de la réalité étant programmé sur l'ordinateur, l'élève peut expérimenter à travers le modèle et ainsi découvrir les règles sous-jacentes aux phénomènes qu'il étudie. Cette approche se révèle très utile dans les cas où l'expérimentation réelle s'avérerait trop dangereuse ou trop coûteuse. L'enseignant veillera cependant à ne pas remplacer par une simulation sur ordinateur les expériences qu'il a la possibilité de réaliser réellement. L'enseignant sélectionnera les parties de cours où l'ordinateur peut lui apporter quelque chose, notamment les lois qui sont introduites sous la forme "Si on faisait telle chose, on pourrait observer que ...". Certaines de ces lois pourraient être introduites par simulation sur ordinateur, le plan d'expérience étant réalisé en classe avec les élèves.

3. POUR L'ORDINATEUR A L'ECOLE, OUI, MAIS COMMENT ?

Les parents, les enseignants et les jeunes ont aujourd'hui, pour la plupart, conscience de ce que l'informatique constitue une branche d'avenir. Mais, à certains moments, ils éprouvent des doutes, ressentent des craintes ou se sentent tout simplement démobilisés par le halo d'ésotérisme qui entoure l'utilisation de l'ordinateur. Il importe donc de démystifier le phénomène.

En ce qui concerne l'enseignement assisté par ordinateur, il est impensable d'envisager que ce type d'enseignement soit appelé un jour à remplacer les enseignants ; au contraire, l'E.A.O. doit faire partie de la globalité du projet éducatif (Toutes les formes d'enseignement peuvent être utiles aux jeunes puisqu'ils forment eux-mêmes des groupes très hétérogènes. On ne peut négliger ni les cours, ni les livres, ni les syllabus, ni les exercices, ni le dialogue collectif).
L'ordinateur n'exclut donc pas la parole du maître. Il en est l'adjoint actif. Entre-eux s'établit une complémentarité indissociable. Ce type de relation nouvelle crée une pédagogie dont l'avantage est d'être motivante. L'élève dispose de la facilité de dialoguer soit avec l'ordinateur, soit avec le maître. L'approche de la matière a pris pour lui une autre dimension : il acquiert chaque nouvelle notion comme une solution à un problème qu'il s'est posé ou que lui a présenté l'enseignant. Pour le maître, il ne s'agit plus d'étaler des solutions mais bien de poser des problèmes. Cela va à l'encontre d'une pédagogie qui, à longueur de trimestres, présente à l'enfant-élève des solutions formalisées à des problèmes qu'il n'a pas eu l'occasion de rencontrer.
Dans un environnement informatisé, l'enfant n'est plus un simple récepteur ou répondeur. Il n'est pas non plus un simple observateur du matériel que le maître manipule. Il devient l'acteur qui apprend en faisant et qui pense ce qu'il fait. Ses qualités cognitives se développent plus harmonieusement et avec de meilleures garanties puisqu'il apprend par l'action.

Tout n'est pourtant pas aussi aisé qu'on pourrait le croire. Un point essentiel de notre réflexion ramène le débat sur le problème de la formation des enseignants. Ceux-ci doivent, en effet, s'adapter à un monde qui, jusqu'il y a peu, leur était inconnu. Déjà de nombreux efforts ont été consentis dans le domaine de la préparation ou de la formation récurrente des éducateurs. Nous sommes malheureusement loin du compte et il faudra réactiver le processus si nous ne voulons pas voir nos pédagogues devenir les élèves de l'ordinateur ou pire encore, ceux de leurs élèves entre-temps devenus maîtres de l'outil.
Un autre élément nous porte également à réfléchir : c'est celui de la construction des programmes adaptés aux matières enseignées - les logiciels.
Un dernier point est également à retenir : les moyens accordés au développement de l'informatique. En effet, l'utilisation de l'ordinateur dans les écoles exige de grands moyens d'investissement en qualification des enseignants et surtout en frais de matériel.

Examinons plus attentivement ces trois problèmes :

1. La formation des enseignants

L'appropriation du savoir par l'élève passe nécessairement par la maîtrise des moyens qui servent à le véhiculer. La première tâche de l'enseignant consiste donc à apprendre à l'élève non seulement des connaissances mais aussi l'utilisation efficace des moyens. Cela suppose de l'éducateur qu'il soit compétent sur le contenu car l'enseignant ne saura remplir sa fonction de médiateur que s'il maîtrise le support du message aussi bien que le contenu lui-même et qu'il participe à la fabrication des supports. En effet, il est prouvé qu'on maîtrise d'autant mieux un type de support si on a participé à son élaboration et à sa production. D'où l'accent à mettre sur l'idée que les enseignants soient formés !

Les objectifs de la formation à donner sont donc de trois ordres :

- Initiation à l'informatique.
  Il faut que les éducateurs apprennent à se servir des moyens informatiques et à réaliser au moins des programmes simples.
- Ouverture aux problèmes liés à l'informatique, problèmes liés au développement de l'informatique comme technique et savoir, problèmes liés à l'informatisation de la société.
- Initiation à l'usage de l'informatique dans la pédagogie et dans la recherche psychologique et pédagogique.

2. La construction des logiciels

L'élaboration d'un matériel est un travail lourd. Le temps nécessaire est fonction de nombreux paramètres : type d'enseignement assisté auquel on veut arriver, connaissance de la matière et de sa pédagogie, outil utilisé, connaissance de l'outil utilisé, ... De toute manière, elle exige des interventions au niveau de la matière, de sa pédagogie et de l'outil utilisé.

Deux agents sont à l'origine de la production de didacticiels : les informaticiens et les enseignants.
Les informaticiens travaillent le plus souvent pour le compte de firmes qui vendent ou louent des didacticiels. Ces produits restent rares sur le marché et sont pour la plupart totalement rigides, ce qui ne convient pas aux besoins des utilisateurs.
Les enseignants souhaitent pouvoir écrire eux-mêmes des programmes adaptés à leur pédagogie, à leurs problèmes. Mais cela nécessite une formation plus complète en informatique et notamment l'apprentissage d'un langage de programmation. Le type de formation variera en fonction de la complexité des problèmes auxquels l'enseignant souhaite s'attaquer. Certains langages (tutor, ego, ...), appelés langages d'auteurs, ont été spécialement conçus pour le développement de didacticiels. Mais ces langages sont extrêmement limitatifs puisqu'ils permettent essentiellement de réaliser des didacticiels de type tutoriel. De plus, si les constructeurs proclament que ces langages ont été spécialement conçus pour des non-informaticiens, en réalité leur apprentissage nécessite une véritable formation.

3. L'investissement

L'informatique à l'école deviendra très rapidement une réalité. Son développement ne fait que commencer et déjà les projets se multiplient : banques de données, banques de questions, séquences de cours ou programmes d'enseignement ... Tout cela suppose la présence dans les locaux scolaires d'un matériel approprié. Ce matériel coûte encore relativement cher malgré les progrès réalisés. Il reste à trouver le moyen de diminuer les coûts. Peut-être qu'en fabriquant les ordinateurs scolaires chez nous, on y arriverait ... Mais cela est une autre réflexion !

4. LA SITUATION BELGE

La réalisation belge la plus marquante consiste en la création, par le Ministère de l'Education nationale, d'un réseau de centres d'information sur l'enseignement et l'apprentissage assisté par ordinateur, le réseau O.S.E. abrégé de Ordinateur au Service de l'Education.

Celui-ci a pour objectifs de :

- réunir et enrichir une documentation, aussi complète que possible, sur la micro-informatique en général et les problèmes que pose son intégration dans la société et à l'école ; sur les logiciels disponibles ; sur la théorie et la pratique de l'E.A.O., ainsi que sur les différentes expériences tentées dans ce domaine dans les écoles belges et étrangères ; sur les différents types de micro-ordinateurs, leurs caractéristiques, les possibilités graphiques, les extensions possibles, les langages de programmation disponibles, etc. ...;

- répondre aux demandes particulières d'information émanant des directions d'écoles, de professeurs, d'associations de parents, sur la micro-informatique, l'achat de matériel, l'élaboration de logiciels, etc. ...;

- organiser des séances d'information, des exposés sur, par exemple, l'expérience de l'utilisation d'un matériel intéressant dans une école de la région, un langage de programmation particulièrement orienté vers une création de logiciels pédagogiques, des didacticiels mis au point par une équipe de professeurs de la région, un logiciel de traitement de texte particulièrement performant;

- présenter du matériel et des logiciels que les enseignants pourront essayer avec l'aide des personnes du centre.

D'autre part, le 18 décembre 1983, parut au Moniteur belge un décret de la communauté française relatif à l'informatique à l'école (cfr appendice 1). Examinons ce qu'il propose quant aux trois problèmes soulevés ci-dessus.

L'article 3 du décret précise que c'est l'Exécutif de la Communauté qui "organise des journées d'études afin d'assurer la formation ou le recyclage des enseignants appelés à enseigner les notions d'informatique ou à utiliser l'ordinateur dans le cadre de leurs cours".
Il incombe donc à la communauté d'organiser ces recyclages ; chaque pouvoir organisateur, au nom de la liberté pédagogique qui lui est reconnue, pourra garder la maîtrise du recyclage de ses enseignants.

En ce qui concerne l'achat de matériel, l'article 4 du décret en parle mais sans préciser par qui cet achat sera effectué. Il précise en effet : "L'achat de matériel se fera en fonction du développement des programmes de formation ou de recyclage des enseignants et de la fabrication des didacticiels ainsi qu'en fonction de l'évolution de la technique de base." Il est évident que l'on ne part pas de rien : beaucoup d'écoles primaires et secondaires ont déjà un équipement informatique. Dans l'enseignement secondaire de transition, le cours d'informatique est d'ailleurs une option complémentaire au troisième degré. Il peut aussi exister dans les options groupées à partir du deuxième degré. Il ressort d'ailleurs d'une enquête effectuée par le groupe de sociologie wallonne de l'UCL sur l'informatique et l'enseignement secondaire que quatre-vingts pour cent des établissements d'enseignement secondaire disposent déjà de configurations informatiques. Cette enquête, basée sur un échantillon représentatif des écoles de l'enseignement secondaire général, photographie la situation à un moment précis du temps : le 15 novembre 1983. A l'heure actuelle, il est plus que probable que cette proportion se soit encore accrue. Cependant, il faut reconnaître que le coût du matériel, jugé trop cher, demeure encore un frein essentiel.
L'Education Nationale, quant à elle, intervient de diverses façons : elle achète du matériel. Celui-ci est placé prioritairement dans les écoles de l'enseignement spécial; elle octroie des subsides à l'enseignement secondaire et supérieur communal, provincial et libre à concurrence de soixante pour cent de la valeur du matériel. (Un crédit spécial de 90 millions a d'ailleurs été accordé à l'Education Nationale pour l'achat de matériel didactique de pointe.)

En ce qui concerne l'élaboration de logiciels, la situation me semble ambiguë. En effet, dans son discours d'ouverture de la Hulpe du Club Athéna, [BERT, 1984], Monsieur Bertouille, Ministre de L'Education Nationale, souligne qu'il est urgent de mettre au point des logiciels pour les écoles. Mais il parle en fait de logiciels facilitant la gestion des établissements scolaires (tel qu'un outil pour la confection des horaires) et non de logiciels d'aide à l'enseignement. Il semble donc que cette distinction, bien que fondamentale, ne soit pas encore bien perçue ...

Le développement de l'E.A.O. est donc encore entravé par divers obstacles au niveau de la diffusion et surtout au niveau de la conception. Espérons que d'ici quelque temps, sous l'influence des centres O.S.E., les décisions prises par l'Education Nationale soient un peu plus hâtives.

CHAPITRE 2

ANALYSE FONCTIONNELLE

1. PROJET

1.1. Présentation générale

Les enseignants sont les premiers concernés en matière d'E.A.O. . Ils connaissent les problèmes et souhaitent disposer d'outils adaptés à leurs besoins. Cependant, comme nous l'avons déjà souligné, le développement d'un didacticiel demande du temps et des connaissances en informatique. Devons-nous demander aux enseignants de devenir informaticiens ou aux informaticiens de devenir pédagogues ? Nous pensons que la collaboration informaticien-enseignant est une solution réaliste au problème. Chacun reste le spécialiste de son domaine. Ce travail s'est réalisé en collaboration avec Monsieur Debrabandère, instituteur d'une classe de quatrième primaire.

Le projet que nous avons élaboré s'adresse à l'enseignement primaire : il s'agit de l'étude des fractions au niveau de la quatrième année.
Cette étude consiste en :

- une introduction du concept par approche graphique et par notation chiffrée;
- une étude des opérations :
    1. additions de fractions de même dénominateur ou l'un multiple de l'autre;
    2. soustractions à résultat positif : soustractions de fractions de même dénominateur ou l'un multiple de l'autre;
    3. multiplications de fractions par un entier positif;
    4. divisions de fractions par un entier positif;
    5. simplifications de fractions : soit par division du numérateur et du dénominateur, soit par réduction à un entier plus une fraction inférieure à l'unité.

1.2. Objectifs

1. Cette étude doit se réaliser progressivement, en tenant compte des difficultés éprouvées par l'élève. Une approche personnalisée s'avère donc nécessaire.

2. Etant donné qu'une partie importante du logiciel consistera en la réalisation d'exercices, l'aspect "discussion" de la réponse est impératif. Le résultat de la "validation" de la solution proposée par l'élève ne peut donc résulter en une seule réponse affirmative ou négative.

3. On désire également qu'une évaluation des élèves soit réalisée. Il ne s'agit pas ici d'attribuer une côte, mais de réaliser un dossier par élève.

4. Comme l'instituteur concerné (Monsieur Debrabandère) souhaiterait utiliser cet outil informatique aussi bien pour faire de l'apprentissage que du rattrapage, on se consacrera, dans le cadre de ce mémoire, à l'apprentissage du point de vue compréhension aussi bien que du savoir-faire.

En effet, ces deux possibilités ayant des approches bien différentes, on se contentera d'étudier la première qui nous semble beaucoup plus intéressante. De plus, rien n'empêche que, par la suite, l'autre dimension soit abordée.

1.3. Qu'est-ce qu'apprendre ?

Comme nous mettrons l'accent sur l'apprentissage, il serait peut-être bon de définir ce que l'on entend par ce terme. (DUFO, 1985)

Si l'on ouvre un dictionnaire français au terme "apprendre", on peut y découvrir deux définitions, correspondant d'ailleurs à deux termes différents dans la langue anglaise :

apprendre =
- s'instruire (learning)
  j'apprends le russe;
- instruire (teaching)
  j'apprends le russe à quelqu'un.

On ne peut cependant pas en rester là, la langue française nous proposant des expressions très révélatrices et très diverses :

Je peux apprendre :

- une mauvaise nouvelle;
- à danser;
- à mes dépens;
- par coeur;
- à vivre.

Force est donc d'approfondir notre analyse.

Pour ce faire, faisons appel à la syntaxe. Elle nous indique quatre directions. En effet :

1. Je peux apprendre que :
   - je peux apprendre qu'il est mort;
   - je peux apprendre que la bourse a monté.

   L'acte d'apprendre est ici un acte d'information.
   Son résultat est le renseignement.

2. Je peux apprendre à :
   - je peux apprendre à danser;
   - je peux apprendre à jouer.

   L'acte d'apprendre est ici un acte d'apprentissage.
   Son résultat est un savoir-faire.

3. Je peux apprendre (verbe intransitif) :
   - je peux apprendre en chantant;
   - je peux apprendre tous les jours.

   L'acte d'apprendre est ici un acte d'étude.
   Son résultat est une compréhension.

4. Je peux également apprendre à être.

On le voit, la question n'est pas simple.
Ce que l'on peut en tout cas affirmer, c'est que :

1. apprendre, c'est acquérir une information, un savoir-faire ou une compréhension;
2. apprendre est un acte que l'on exerce sur soi-même;
3. on peut apprendre sans enseignant et même sans enseignement (j'apprends ma langue). C'est l'éducation spontanée, par contraste à l'éducation intentionnelle (l'école);
4. apprendre n'est le corrélat d'enseigner qu'à deux conditions :
   - être soumis à un enseignement;
   - cet enseignement doit avoir atteint son but.

Mais si faire apprendre n'est pas obligatoirement le résultat de l'enseignement, il en est nécessairement le but, du moins si l'enseignement veut sa propre réussite. C'est d'ailleurs ce que suggère l'étymologie : "insignare", montrer, indiquer. Cette fin n'est pas toujours atteinte; il n'en reste pas moins que c'est elle qui donne son sens à l'acte d'enseigner. Ce qui appelle deux précisions.

D'abord, il faut que la fin soit consciente. Je puis découvrir par hasard ou par ruse une information que quelqu'un voulait me cacher; j'ai donc appris de lui - comme le psychanaliste "apprend" de son patient - mais il ne m'a pas enseigné. Ainsi "apprend-on" les desseins de l'ennemi, ou les techniques d'un concurrent. Je puis apprendre par ouï dire ce qu'enseigne un collègue; personne ne dira qu'il m'a enseigné ! L'intention de faire apprendre est inhérente à l'activité d'enseigner.

Ensuite, "faire apprendre" n'a pas le même sens que "faire savoir". Faire savoir à quelqu'un que sa rue est la troisième à gauche, ou qu'il a échoué à son examen, n'est pas l'enseigner mais le renseigner. Enseigner est une activité qui vise à susciter une activité. Ceux qui réduisent l'enseignement à une transmission de savoirs le méconnaissent totalement. La plupart des innovations ont l'avantage de nous montrer qu'enseigner n'est pas enregistrer, mais faire faire : leur but n'est pas d'enseigner moins, mais d'enseigner mieux, même si ce mieux se paie d'un moins dans le domaine de l'autorité ou du programme. En tout cas, elles font ressortir qu'"apprendre" est un verbe actif.

2. APPLICATIONS

2.1. Critère d'identification

Une application est un traitement quasi autonome par rapport aux autres applications du projet. Les applications ne communiquent entre elles que par l'échange d'agrégats d'information. (BODA et alt., 1983)

2.2. Les différentes applications

2.2.1. Etude et compréhension des fractions

Description

Objectifs

Cette application vise à aider des élèves de quatrième primaire à comprendre la notion de fractions. Ce nouveau concept sera d'abord introduit par une approche graphique, par la notation chiffrée ensuite. Une fois ce concept assimilé, une étude de quelques opérations simples (addition, soustraction, division, multiplication, simplification) sera réalisée.

Règles

- Dans le cadre de ce mémoire, on se limitera, quant à la matière à enseigner, au programme de la quatrième année primaire. Toutefois, une extension doit s'avérer possible.
- On suppose que les élèves de quatrième année primaire n'ont aucune notion des fractions.
- Cette étude nécessite un suivi progressif de l'enfant : en effet, si la matière abordée lui paraît trop simple ou trop difficile, l'élève s'en désintéressera.

Découpe en phases

.Choix d'un exercice.
.Présentation et proposition de cet exercice.
."Examen" de la réponse de l'élève.

2.2.2. Evaluation de l'élève

Description

Objectifs

Cette application vise à réaliser un dossier de chaque élève, ainsi qu'un historique de celui-ci.

Règles

- Ce n'est pas une côte que l'on désire mais un véritable dossier contenant les erreurs commises par l'élève, ses facilités et ses difficultés, ...

Découpe en phases

.Identification de l'élève.
.Enregistrement et mise à jour des dossiers sur les élèves.
.Consultation de ces dossiers.

2.3. Justificatif de la découpe

La première de ces applications est relative au flux "enseignement" : du choix d'un exercice à l'obtention d'une réponse satisfaisante. La seconde application concerne le flux "évaluation" : de l'enregistrement d'informations relatives aux élèves à l'étude de ces renseignements (étude statistique ou autre). Ces deux applications sont quasi autonomes car elles ne communiquent entre elles que par l'échange d'agrégats (cfr figure 2.1) : après la réalisation d'un exercice, si l'élève a éprouvé des difficultés à le résoudre, un message relatif aux erreurs commises par celui-ci est communiqué à l'application "Evaluation de l'élève"; inversement, avant le choix d'un nouvel exercice à soumettre à l'élève, un message relatif à ses difficultés et à ses facilités est communiqué à l'application "Etude et compréhension des fractions" en vue d'un meilleur encadrement de l'étudiant.

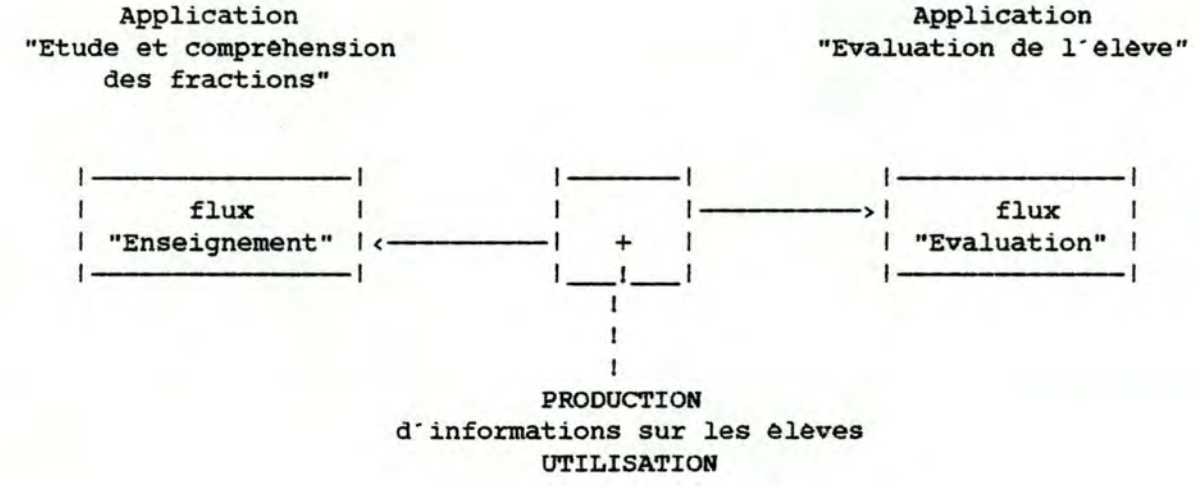


Figure 2.1

3. PHASES

3.1. Critère d'identification

Une phase est un traitement quasi autonome possédant une unité spatio-temporelle d'exécution. Cette unité d'exécution implique que la phase soit entièrement exécutée dans une cellule d'activités, c'est à dire un centre d'activités homogènes dans le temps et dans l'espace, doté de ressources humaines et/ou matérielles et pourvu de règles de comportement nécessaires à son fonctionnement. (BODA et alt., 1983)

3.2. Les différentes phases

3.2.1. Choix d'un exercice

Description

Objectifs

Il s'agira ici de choisir un exercice à proposer à l'élève en tenant compte de ses aptitudes, aussi bien positives que négatives.

Règles

- Rappelons que l'étude de la notion de fractions nécessite une approche personnalisée.
- Il faudra éviter d'ennuyer l'élève par des exercices trop faciles ou trop compliqués, ou par des exemples trop peu variés.

Découpe en fonctions

. Choix d'un type d'exercice.
. Suivi du déroulement d'un type d'exercice.

3.2.2. Présentation et proposition d'un exercice

Description

Objectifs

Il s'agira de présenter clairement l'exercice à résoudre, en utilisant toutes les ressources disponibles de la machine (couleurs, graphisme en haute résolution ...).

Règles

- L'exercice sera présenté clairement à l'élève, évitant toute ambiguïté.
- Le problème posé s'énoncera clairement en utilisant un vocabulaire propre à un enfant de dix ans.

Découpe en fonctions

. Génération de données.
. Présentation de l'énoncé du problème à résoudre.
. Affichage des différentes figures.
. Partage des figures en parties équivalentes.
. Coloriage des parties équivalentes.

3.2.3. "Examen" de la réponse de l'élève

Description

Objectifs

On désire que la réponse de l'élève ne soit pas seulement considérée comme correcte ou non, mais qu'un véritable examen soit réalisé. En effet, si le résultat obtenu par l'élève est correct, on peut supposer qu'il a surmonté le problème à résoudre, mais peut-on affirmer qu'il a compris ?
D'autre part, si ce résultat est faux, peut-on prétendre que l'élève n'a rien assimilé du tout ?

Règles

- Il ne faut pas seulement "critiquer" la réponse de l'élève, mais son raisonnement.

Découpe en fonctions

. Enregistrement du raisonnement de l'élève.
. Calcul de la réponse au problème posé.
. Validation de la réponse de l'élève.
. Mise à jour des renseignements sur l'élève.
. Production d'explications.

3.2.4. Mise à jour du dossier de l'élève

Description

Objectifs

Comme nous l'avons déjà dit, nous conserverons un compte-rendu de l'état d'avancement du travail de chaque élève. Cette phase gérera donc l'historique des différents élèves et ceci, en collaboration avec la première application.

Règles

- Le contenu du dossier de chaque élève devra contenir au minimum les informations suivantes : le nom et le prénom de l'élève, sa classe et son âge, s'il est doubleur ou non, les dates auquelles il a travaillé, les erreurs qu'il a commises le plus fréquement et l'évolution de son étude.
  Il faudra également prévoir l'extensibilité de ces dossiers, les caractéristiques reprises ci-dessus ne correspondant qu'aux informations minimales à conserver.

Découpe en fonctions

. Création d'un dossier.
. Mise à jour d'un dossier.

3.2.5. Consultation de l'éducateur

Description

Objectifs

Il s'agira ici de permettre à l'enseignant de consulter les dossiers des différents élèves. Cette consultation s'effectuera, soit via la console, soit par l'impression des renseignements demandés.

Règles

- La présentation des renseignements demandés se veut claire et précise, évitant toute ambiguïté.
- Les possibilités de consultation devront être multiples. C'est ainsi que, par exemple, il faudra que l'éducateur puisse obtenir tous les renseignements sur chaque élève; qu'il puisse consulter le dossier d'un élève en particulier; qu'il puisse accéder aux caractéristiques des élèves qui ont travaillé à une date déterminée; qu'il puisse obtenir des statistiques sur les "types d'erreurs" les plus fréquemment commises ou qu'il puisse connaître les élèves qui ont déjà effectué un exercice particulier.
- Il est également primordial de réserver l'accès à ses informations aux seuls enseignants.

Découpe en fonctions

. Consultation via la console.
. Consultation via un listing.

3.2.6. "Examen" de la réponse de l'élève

Description

Objectifs

Comme l'approche de l'étude des fractions se veut personnalisée, le système aura la possibilité de consulter les dossiers des élèves afin de se "remémorer" les caractéristiques de ces derniers, c'est à dire l'état d'avancement de leur étude, les facilités et les difficultés qu'ils ont rencontrées, les erreurs qu'ils ont commises le plus fréquemment ...

4. FONCTIONS

4.1. Critère d'identification

Une fonction correspond au niveau élémentaire de la nomenclature des traitements ; elle est associée à un objectif et à un comportement organisationnel considérés comme élémentaires par l'organisation. Elle possède une sémantique simple. (BODA et alt., 1983)

4.2. Les différentes fonctions

4.2.1. Choix d'un type d'exercice

Description

Cette fonction a pour but de déterminer le type d'exercice le plus approprié à l'état actuel des connaissances de l'élève. Cette fonction prendra donc en considération l'historique de celui-ci.

4.2.2. Suivi du déroulement d'un exercice

Description

Cette fonction traite de la gestion d'un type d'exercice. En effet, nous avons vu qu'une génération de données représentait une occurence du type d'exercice; et, d'autre part, un type d'exercice comptera plusieurs occurences. Cette fonction arbitrera donc ce nombre d'occurences.

4.2.3. Génération de données

Description

Lorsqu'un type d'exercice a été choisi, il faut alors générer des données, données qui représentent une occurrence de ce type d'exercice. C'est le rôle de cette fonction.

4.2.4. Présentation de l'énoncé du problème à résoudre

Description

Cette fonction consiste à présenter clairement à l'élève l'objectif à atteindre pour résoudre le problème.

4.2.5. Affichage des différentes figures

Description

Cette fonction consiste dans l'affichage de différentes figures (cercle, rectangle, polygône régulier) qui fourniront un appui graphique en vue d'une meilleure compréhension du problème à résoudre.

4.2.6. Partage des différentes figures en parties équivalentes

Description

Cette fonction traite du partage des différentes figures précitées en parties équivalentes. Cela permettra ainsi aux élèves de mieux visualiser, sur des exercices concrets, la notion de fraction.

4.2.7. Coloriage des parties équivalentes

Description

Cette fonction consiste en une mise en relief de certaines parties équivalentes dessinées dans une figure. Cela permettra notament à l'élève de se familiariser avec la notion de numérateur d'une fraction.

4.2.8. Enregistrement du raisonnement de l'élève

Description

Etant donné que la critique de la réponse de l'élève ne peut seulement s'opérer sur sa réponse finale, cette fonction consistera en l'enregistrement des résultats intermédiaires et de la solution à un problème donné.

4.2.9. Calcul de la réponse au problème posé

Description

Afin de pouvoir porter un jugement sur le resultat d'un exercice obtenu par un élève, il faut préalablement calculer soi-même la solution de cet exercice. C'est le but de cette fonction.

4.2.10. Validation de la réponse de l'élève

Description

Cette fonction consiste à détecter éventuellement l'erreur ou les erreurs de raisonnement commise(s) par l'élève.

4.2.11. Mise à jour des renseignements sur l'élève

Description

Cette fonction consiste en la mise à jour de l'historique de l'élève. Il s'agira soit d'une confirmation de la compréhension de l'élève, soit d'une infirmation concernant ce type d'exercice.

4.2.12. Production d'explications

Description

Dans le cas où l'élève n'admet pas son erreur, ou ne la comprend pas, une résolution détaillée du problème lui sera communiquée. C'est le rôle de cette fonction.

4.2.13. Création d'un dossier

Description

Cette fonction interviendra lorsqu'un élève inconnu du système vient travailler. Un dossier sera alors ouvert à son nom.

4.2.14. Mise à jour des dossiers

Description

Cette fonction traitera de la gestion des dossiers des différents élèves. Elle consistera en la sélection d'un dossier, sa mise à jour en fonction du travail effectué par l'étudiant ...

4.2.15. Consultation via la console

Description

Cette fonction permettra à l'éducateur de consulter les renseignements concernant l'ensemble des élèves. Cette consultation interactive lui permettra de collecter une foule d'informations par un jeu habile de questions - réponses.

4.2.16. Consultation via un listing

Description

Cette fonction permettra également à l'enseignant de consulter les renseignements concernant l'ensemble des élèves. Elle fera l'objet d'une utilisation moins fréquente que la consultation via une console et sera surtout utile pour la production d'états périodiques caractérisant les étapes de la progression de l'étude des élèves.
Cette fonction sera également utilisée lorsque l'éducateur désire beaucoup de renseignements, comme par exemple l'ensemble des informations concernant tous les élèves.

CHAPITRE 3

CONCEPTION DES DIFFERENTS MODULES

Nous avons découpé notre projet en différents modules de complexité raisonnable; nous allons maintenant étudier la conception de ces modules et nous en verrons l'implémentation lors du chapitre suivant.
Nous ne détaillerons cependant dans ces deux chapitres que les modules les plus intéressants, ceux que nous avons écartés étant jugés trop simples.

1. AFFICHAGE DES DIFFERENTES FIGURES

1.1. Spécifications concrètes

Ce module, destiné à afficher différentes figures à l'écran, sera déclenché lors de la présentation à l'élève d'un exercice nécessitant un support graphique. Comme ce module ne fera pas l'objet d'un programme en tant que tel, les vérifications concernant les valeurs des paramètres ne seront pas prises en considération.
En effet, tous les paramètres d'entrée fournis à ce module seront transmis par programme et leur valeur sera supposée cohérente; étant donné, par exemple, la longueur du côté d'un carré et les coordonnées de son centre, le module ne vérifiera donc pas s'il est possible d'afficher la figure entièrement ou partiellement à l'écran.

1.2. Paramètres

Entrée :

- TYPESUR : (type de la surface à afficher)
  Ce paramètre numérique désigne la figure à afficher à l'écran; sa valeur comprise entre 1 et 9 désignera respectivement le carré, le carré sur pointe, le rectangle horizontal, le rectangle vertical, le cercle, le triangle équilatéral, l'hexagone, l'octogone et le segment.

- COORD1 : (1ère coordonnée)
  Ce paramètre numérique représente la coordonnée-abscisse du centre de la figure à afficher.

- COORD2 : (2ème coordonnée)
  Ce paramètre numérique représente la coordonnée-ordonnée du centre de la figure à afficher.

- PARAM1 : (1er paramètre)
  Ce paramètre numérique représente le premier ou le seul paramètre caractérisant la figure à afficher. Dans le cas d'un carré, d'un rectangle, d'un triangle équilatéral, d'un hexagone ou d'un octogone, il indiquera la longueur d'un coté; il représentera le rayon d'un cercle ou exprimera la longueur d'un segment.

- PARAM2 : (2ème paramètre)
  Ce paramètre numérique représente éventuellement le second paramètre caractérisant la figure à afficher. En ce qui concerne les neuf possibilités présentes, il interviendra uniquement pour indiquer la largeur d'un rectangle.

Sortie :

- ABSC : (abscisse)
  Ce tableau numérique contiendra les abscisses du centre et des sommets de la figure. Les valeurs qu'il comportera seront évaluées lors de l'affichage, excepté en ce qui concerne l'abscisse du centre de la figure puisqu'il est fourni comme paramètre d'entrée.
  L'utilité de ce tableau sera expliquée lors de l'étude du module "Partage des figures en parties équivalentes".

- ORDO : (ordonnée)
  Ce tableau numérique contiendra les ordonnées du centre et des sommets de la figure.
  Son utilité sera également expliquée par la suite.

2. PARTAGE DES DIFFERENTES FIGURES EN PARTIES EQUIVALENTES

2.1. Spécifications concrètes

Ce module, dédicassé à partager différentes figures en parties équivalentes, sera déclenché lors de la présentation à l'élève d'un exercice nécessitant un support graphique. En ce qui concerne la vérification de la cohérence de la valeur des paramètres, ce module réagira comme le précédent; si on le considère donc comme un programme, un léger désagrément pourra survenir lorsque, par exemple, la longueur du côté d'un quadrilatère n'est pas divisible par le nombre de droites verticales ou horizontales nécessaires au découpage de la surface. Je m'explique .... Si la longueur du carré que je dois découper en six parties équivalentes (soit en traçant deux droites verticales et trois horizontales) est fixée à cinquante, les trois parties obtenues en coupant le carré selon les droites horizontales ne seront pas identiques puisque cinquante n'est pas divisible par trois.

Cette mise au point étant faite, précisons maintenant ce que nous entendons par "parties équivalentes". En raison des sollicitations de l'instituteur concerné (Monsieur Debrabandère), nous limiterons cette notion à des parties identiques de part leur forme (Figure 3.1). Les figures A et B seront donc acceptées tandis que les découpes C et D seront exclues.

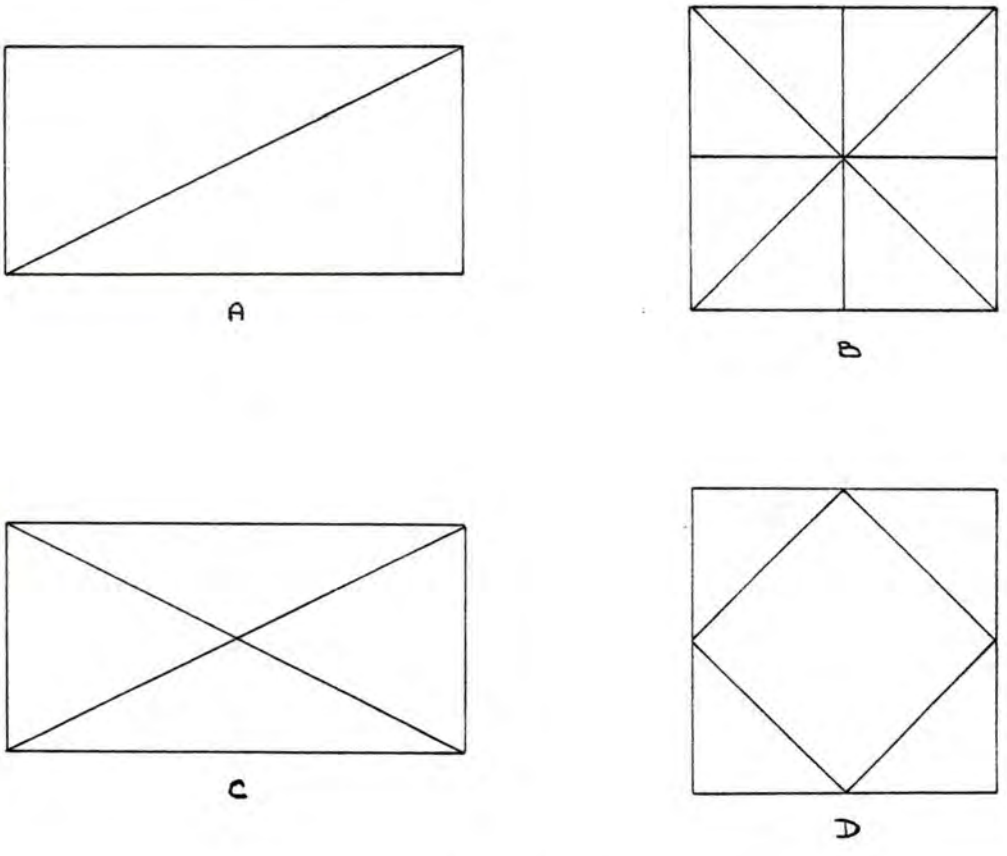


Figure 3.1

2.2. Paramètres

Entrée :

- TYPESUR : (type de la surface)
  Ce paramètre numérique désigne la figure à partager en parties équivalentes; sa valeur comprise entre 1 et 9 désignera respectivement le carré, le carré sur pointe, le rectangle horizontal, le rectangle vertical, le cercle, le triangle équilatéral, l'hexagone, l'octogone et le segment.

- PARAM1 : (1er paramètre)
  Ce paramètre numérique représente le premier ou le seul paramètre caractérisant la figure à partager en parties équivalentes. Dans le cas d'un carré, d'un rectangle, d'un triangle équilatéral, d'un hexagone ou d'un octogone, il indiquera la longueur d'un coté; il représentera le rayon d'un cercle ou exprimera la longueur d'un segment.

- PARAM2 : (2ème paramètre)
  Ce paramètre numérique représente éventuellement le second paramètre caractérisant la figure à partager en parties équivalentes. En ce qui concerne les neuf possibilités considérées,il interviendra uniquement pour indiquer la largeur d'un rectangle.

- NBREPAR : (nombre de parties à dessiner)
  Ce paramètre numérique représente le nombre de parties équivalentes à dessiner dans la figure. Les possibilités retenues sont deux, trois, quatre, cinq, six, huit, dix et douze.

- DIFFPAR : (difficulté du partage)
  Ce paramètre numérique représente la difficulté lors du partage des figures. Il y a en effet plusieurs manières de diviser une figure en parties équivalentes.
  Ce paramètre variera de 1 à 4. Cependant, les quatre possibilités n'existeront pas nécessairement étant donné une figure et un nombre de parties équivalentes à dessiner.

- ABSC : (abscisse)
  Ce tableau numérique contient les abscisses du centre et des sommets de la figure. Les valeurs qu'il comporte ont été évaluées lors de l'affichage.

- ORDO : (ordonnée)
  Ce tableau numérique contient les abscisses du centre et des sommets, de la figure. Les valeurs qu'il comporte ont été évaluées lors de l'affichage.

3. COLORIAGE DES DIFFERENTES PARTIES EQUIVALENTES DES FIGURES

3.1. Spécifications concrètes

Ce module, déclenché également lors de la présentation à l'élève d'un exercice nécessitant un support graphique, mettra en relief un certain nombre de parties équivalentes dessinées dans la figure.

Pour attirer l'attention sur certaines parties équivalentes, plusieurs possibilités nous étaient offertes : les hachurer, les marquer d'un caractère quelconque ou les colorier. Ayant testé et comparé ces différentes techniques, la dernière nous sembla la plus agréable. Cependant, afin d'éviter toute confusion, certaines précautions ont dû être prises.... En effet, examinons les trois découpes suivantes (Figure 3.2) : la première et la troisième représentent clairement les fractions "1/4" et "3/4", tandis que la seconde peut prêter à confusion : selon certains élèves, elle représente "2/4", selon d'autres, "1/2".

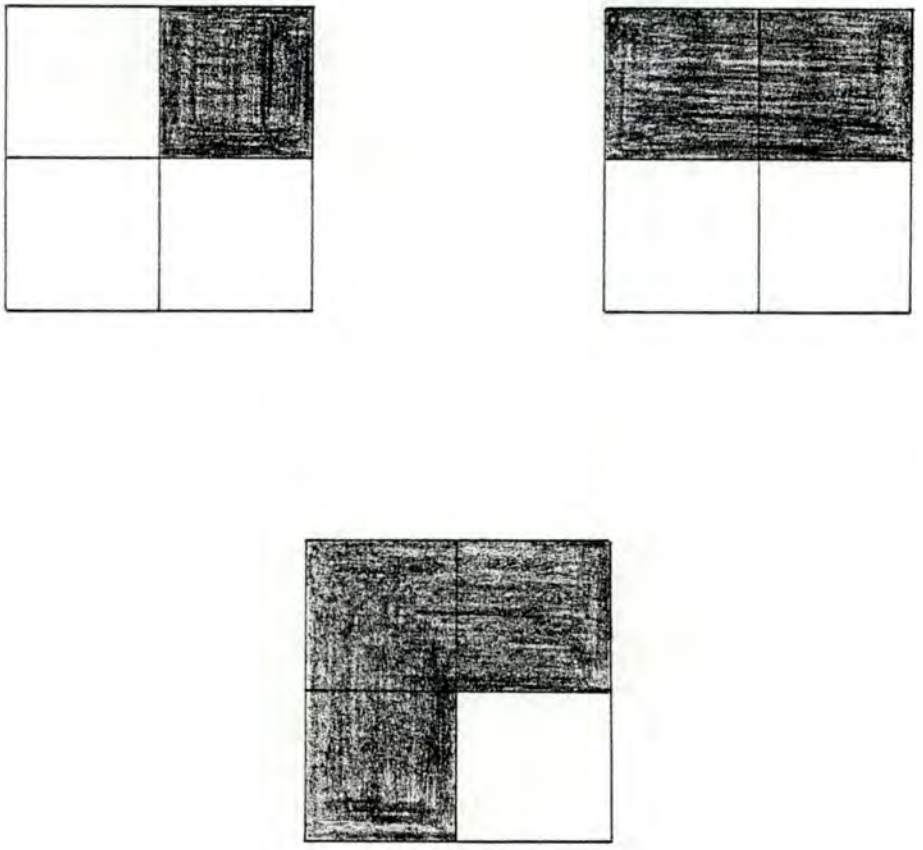


Figure 3.2

La deuxième affirmation, bien qu'exacte et identique à la première, ne signifie pas pour autant que l'étudiant a bien compris l'équivalence des deux solutions. En effet, dans ce cas, l'élève a très bien pu identifier deux parties dessinées dans la figure dont l'une était coloriée, et en déduire que la fraction représentée valait "1/2".
Par contre, l'élève qui propose la fraction "1/2" pour la découpe de la (Figure 3.3) me prouve, quant à lui, qu'il a bien assimilé l'équivalence de ces deux solutions.

Cette remarque peut nous paraître assez ridicule, à nous qui maîtrisons cette matière depuis bien longtemps; cependant, lorsque nous avons simulé notre logiciel avec différents élèves, cette confusion est apparue si flagrante qu'elle nous a obligé à réagir.

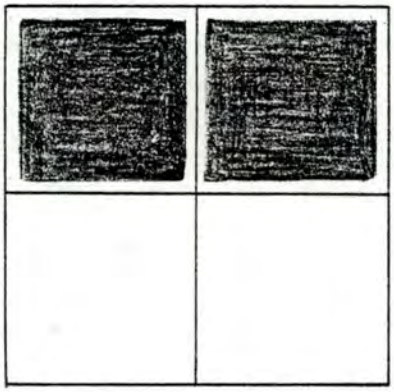


Figure 3.3

3.2. Paramètres

Entrée :

- TYPESUR : (type sur la surface)
  Ce paramètre numérique désigne la figure partagée en parties équivalentes dont on va hachurer certaines parties; sa valeur comprise entre 1 et 9 désignera respectivement le carré, le carré sur pointe, le rectangle horizontal, le rectangle vertical, le cercle, le triangle équilatéral, l'hexagone, l'octogone et le segment.

- PARAM1 : (1er paramètre)
  Ce paramètre numérique représente le premier ou le seul paramètre, caractérisant la figure partagée en parties équivalentes dont on va hachurer certaines parties. Dans le cas d'un carré, d'un rectangle, d'un triangle équilatéral, d'un hexagone ou d'un octogone, il indiquera la longueur d'un coté; il représentera le rayon d'un cercle ou exprimera la longueur d'un segment.

- PARAM2 : (2ème paramètre)
  Ce paramètre numérique représente éventuellement le second paramètre caractérisant la figure partagée en parties équivalentes dont on va hachurer certaines parties. En ce qui concerne les neuf possibilités considérées, il interviendra uniquement pour indiquer la largeur d'un rectangle.

- NBREPAR : (nombre de parties dessinées)
  Ce paramètre numérique correspond au nombre de parties équivalentes dessinées dans la figure. Les possibilités retenues sont deux, trois, quatre, cinq, six, huit, dix et douze.

- DIFFPAR : (difficulté de partage)
  Ce paramètre numérique indique la stratégie suivie lors du partage de la figure en parties équivalentes. Ce paramètre variera de 1 à 4. Cependant, les quatre possibilités n'existeront pas nécessairement étant donné une figure et un nombre de parties équivalentes à dessiner.

- ABSC : (abscisse)
  Ce tableau numérique contient les abscisses du centre et des sommets de la figure. Les valeurs qu'il comporte ont été évaluées lors de l'affichage.

- ORDO : (ordonnée)
  Ce tableau numérique contient les abscisses du centre et des sommets, de la figure. Les valeurs qu'il comporte ont été évaluées lors de l'affichage.

- NBCOLOR : (nombre de parties à colorier)
  Ce paramètre numérique indique le nombre de parties équivalentes à colorier dans la figure.

4. GENERATION DE DONNEES

4.1. Spécifications concrètes

Ce module sera déclenché chaque fois qu'une occurence d'un type d'exercice est présentée à un élève.
Les caractéristiques de ce module seront doubles : il devra s'assurer de l'uniformité des valeurs générées par rapport à un intervalle fixé. Il devra tenir compte des connaissances de l'élève et générer seulement des données adaptées à son savoir.

4.2. Paramètres

Entrée :

- DIFF6 : (difficulté 6)
  Nous comprendrons mieux la signification du nom de ce paramètre après la lecture de ce chapitre.
  La valeur de ce paramètre numérique est comprise entre 1 et 15 et désignera respectivement la génération d'une fraction inférieure à l'unité dont le numérateur vaut un; une fraction quelconque inférieure à l'unité; deux fractions de même dénominateur; deux fractions de même numérateur; deux fractions dont l'une a son dénominateur multiple de l'autre; une fraction équivalente à un naturel; une fraction supérieure à l'unité; une fraction simplifiable; deux fractions de même dénominateur inférieures à l'unité; un naturel et une fraction; deux naturels et une fraction; un naturel et deux fractions; deux naturels et deux fractions; un naturel et une fraction; une fraction et un naturel.

- DIFF7 : (difficulté 7)
  La valeur de ce paramètre numérique prend en considération le type d'opération à effectuer sur les fractions. Sa valeur est déterminante dans certains cas comme par exemple, la soustraction de deux fractions de même dénominateur. Dans ce cas, le numérateur de la première fraction doit en effet être supérieur ou égal à celui de la deuxième.

- DIFF2 : (difficulté 2)
  Les valeurs de ce tableau numérique représentent les huit dénominateurs étudiés, à savoir le demi, le quart, le tiers, le sixième, le huitième, le cinquième, le dixième et le douzième.

- NIV2 : (niveau 2)
  La valeur de ce paramètre numérique désigne un indice du tableau précédent. Il représente le niveau actuel d'assimilation de l'étudiant en ce qui concerne l'étude des dénominateurs de fractions.

Sortie

- NAT1 : (1er naturel)
  Ce paramètre numérique contiendra éventuellement la valeur générée du premier naturel. En effet, certaines fractions à générer peuvent comporter un naturel, d'autres pas.

- NAT2 : (2ème naturel)
  Ce paramètre numérique contiendra éventuellement la valeur générée du deuxième naturel. En effet, il ne faut pas toujours générer deux fractions pour offrir des exercices à l'élève, une seule peut suffire dans certains cas (par exemple, lors d'un exercice concernant la simplification des fractions).

- DENOM1 : (1er dénominateur)
  Ce paramètre numérique contiendra la valeur générée du dénominateur de la première fraction.

- DENOM2 : (2ème dénominateur)
  Ce paramètre numérique contiendra éventuellement la valeur générée du dénominateur de la deuxième fraction.

- NUM1 : (1er numérateur)
  Ce paramètre numérique contiendra la valeur générée du numérateur de la première fraction.

- NUM2 : (2ème numérateur)
  Ce paramètre numérique contiendra éventuellement la valeur générée du numérateur de la deuxième fraction.

5. "EXAMEN" DE LA REPONSE DE L'ELEVE

5.1. Introduction

Comme nous l'avons déjà fait remarquer, le résultat de la validation de la solution proposée par l'élève ne peut résulter en une seule réponse positive ou négative. En effet, si le résultat de l'exercice obtenu par l'enseignant correspond à celui de l'élève, on peut supposer que celui-ci a compris. Mais si sa réponse ne convient pas, lui faire remarquer et proposer un nouvel exercice ne semble pas la meilleure démarche. Il faudrait plutôt détecter l'erreur et expliquer à l'élève ce qu'il a fait et ce qu'il aurait dû faire.

5.2. Comment réaliser un suivi du raisonnement de l'élève ?

Afin de pouvoir "suivre" le raisonnement de l'élève, il est donc nécessaire de connaître la démarche à effectuer pour résoudre le problème ainsi que les erreurs pouvant être commises par celui-ci.
En effet, supposons que l'exercice proposé consiste en l'addition des deux fractions suivantes : "1/4 + 2/4" . Si la réponse de l'élève vaut "3/8", et si nous savons qu'une des erreurs les plus fréquemment commises concernant ce type d'exercice consiste en l'addition des deux numérateurs des fractions et de leur dénominateur, il est alors aisé d'identifier cette erreur typique. D'autre part, si nous savons comment additionner deux fractions de même dénominateur, nous pourrons également expliquer à l'élève ce qu'il aurait dû faire, en s'appuyant au besoin sur un support graphique.

Malheureusement, s'il est facile d'apprendre la démarche à suivre pour résoudre de tels problèmes, il n'en est pas de même en ce qui concerne les erreurs commises par les élèves.
Si, lors d'un exercice concernant la reconnaissance du dénominateur d'une fraction, on propose à l'élève un rectangle divisé en quatre parties équivalentes ; et si la question posée est relative à la représentation d'une partie équivalente par rapport à la figure dessinée, que peut-on déduire de la réponse "1/3" ? Nous savons en effet qu'il suffit de compter le nombre de parties équivalentes représentées, mais nous sommes incapables de "comprendre" le raisonnement de l'élève qui a abouti à cette solution erronée.

Nous ne pourrons donc pas toujours réaliser un suivi du raisonnement de l'élève. Par conséquent, dans le cas présent, nous nous limiterons aux exercices concernant les opérations sur les fractions (addition, soustraction, multiplication, division). Mais même pour ceux-ci, relever les différentes erreurs pouvant être commises par les élèves ne fut pas une tâche aisée.
En effet, malgré nos nombreuses démarches auprès d'instituteurs, nous n'avons pu identifier que quelques erreurs typiques. Comme cela n'était pas suffisant, nous avons donc réalisé une "maquette" comptant soixante-cinq exercices, que nous avons distribuée dans les écoles primaires de la région de Tournai. Plusieurs institutions nous ont répondu et nous avons pu réunir cent dix-sept copies de cette maquette, ce qui représente donc plus de sept mille cinq cents exercices résolus. Il nous resta alors à dépouiller les solutions erronées de ces exercices et à essayer de

comprendre le raisonnement fautif suivi, ce qui nous permit de détecter une cinquantaine de "types d'erreurs".
Ainsi armé, nous estimons que nous pouvons donc suivre, d'une manière satisfaisante, le raisonnement des élèves en nous basant non seulement sur leur réponse finale, mais aussi sur leurs résultats intermédiaires éventuels.

5.3. Comment faire appliquer cette démarche par une machine ?

Maintenant que nous connaissons les deux conditions préalables à la réalisation du suivi du raisonnement de l'élève, examinons leurs implications en terme de programmation.

La première condition suppose que l'on puisse appliquer un raisonnement mathématique à un problème précis; étant donné un exercice concernant les opérations sur les fractions, il faut pouvoir le résoudre et fournir, si nécessaire, les différentes étapes intermédiaires empruntées.

La seconde condition exige la détection de l'erreur ou des erreurs commises dans un raisonnement mathématique. Etant donné le résultat erroné d'un exercice, il faut pouvoir détecter l'origine et la cause de l'erreur. Formulons à ce sujet deux remarques : l'élève peut avoir commis plusieurs erreurs consécutives dans son raisonnement; et d'autre part, rien ne l'oblige à fournir les étapes intermédiaires de sa recherche de la solution. Il peut en effet seulement se contenter de fournir la réponse finale.

Dans une programmation classique, la conséquence de la première condition est facilement réalisable : les démarches à appliquer pour résoudre les exercices précités sont très algorithmiques. Notons cependant, une certaine redondance dans les étapes successives de celles-ci, due à l'analogie des raisonnements à suivre pour traiter les différents problèmes.

Par contre, l'exigence de la deuxième condition est moins simple à réaliser. Seule une suite de "IF" imbriqués semble convenir. Malheureusement, cette solution peu élégante ne prend pas en considération le facteur temps, lequel est primordial dans notre cas (en effet, si le "type d'erreur" commis est le cinquantième dans la liste, il faudra passer en revue les quarante-neuf premiers cas avant d'y parvenir). D'autre part, cette structure de "IF" imbriqués deviendra encore plus inefficace si on tient compte d'une des remarques formulées ci-dessus, à savoir que l'élève a pu commettre plusieurs erreurs consécutives dans son raisonnement. De plus, la liste des "types d'erreurs" recensée jusqu'à présent n'a pas la prétention d'être exhaustive, et cette structure n'est pas apte à en rajouter sans l'intervention du programmeur.

C'est pourquoi, l'approche de l'intelligence artificielle nous semble préférable ...

5.4. Un système expert en E.A.O. ?

5.4.1. Rappels sur les systèmes experts

Un système expert est composé de trois modules : un interface, un système cognitif et une base de connaissances.

1. L'interface fournit la communication homme-machine et permet l'acquisition des données sous différentes formes.

   Il est composé des trois modules suivants :

   - L'interface utilisateur qui permet d'accepter les données et de fournir les résultats, les explications, les questions souvent en langue quasi naturelle.

   - Le module acquisition de connaissances, utilisé par l'expert pour rentrer les connaissances du domaine. Associé à ce module, existe le module de vérification de ces informations, quelquefois limité au contrôle de la syntaxe mais le plus souvent comportant les tests de cohérence avec les connaissances déjà enregistrées.

   - L'interface d'acquisition de données est plus conventionnel que les deux autres. Il est similaire à ceux de la plupart des systèmes interactifs et comprend des facilités pour entrer les données et paramètres du problème ainsi que pour répondre aux questions du système. Il contient en outre le mécanisme pour accéder aux fichiers de données. La plupart des fonctions qui constituent l'interface d'acquisition des données existent déja dans le système interactif qui supporte le système expert.

2. Le système cognitif est l'élément actif du système expert. Il contient un mécanisme de résolution de problèmes (raisonnement ou inférence) et l'applique à la base de connaissances pour répondre aux questions ou pour résoudre le problème posé par l'utilisateur. Pour ce faire, il contient un certain type de connaissances sur la façon de raisonner.
   Une deuxième fonction du système cognitif est celle d'expliquer son processus de raisonnement si l'utilisateur en fait la demande.
   Et enfin, le système cognitif doit aussi fournir les outils nécessaires pour l'acquisition de nouvelles connaissances, la modification de celles existantes et la suppression de celles erronées ou inutiles. Ces altérations de la base de connaissances doivent être faites par un "expert" et non par l'utilisateur.
   En conclusion, le système cognitif est l'élément actif de contrôle d'un système expert qui dirige les activités de résolution de problèmes, explique le comportement du système à la demande de l'utilisateur et gère la base de connaissances.

3. La base de connaissances doit contenir, dans un système idéal, toutes les connaissances du domaine nécessaires pour qu'un système déductif agisse comme un expert.
   Dans la plupart des systèmes experts, un effort est réalisé pour que la base de connaissances soit indépendante du système cognitif et ceci afin de pouvoir changer le domaine d'utilisation du système sans en modifier le mécanisme de raisonnement.

5.4.2. Explicitation de ces concepts dans notre cas

1. Les différents composants principaux d'un système expert ayant été rappelés, étudions maintenant leur signification au sein de notre application. Nous expliciterons d'abord le concept de base de connaissances, celui de système cognitif ensuite et enfin celui d'interface-utilisateur.

   Un raisonnement se compose d'un certain nombre d'étapes. Chacune d'elle peut être représentée sous la forme

   si ‹condition 1› et ‹condition 2› et ... et ‹condition n›

   alors ‹action 1› et ... et ‹action n›,

   que nous appelerons "règle de production".

   Dans le cas de l'addition de deux fractions de même dénominateur, nous aurons par exemple une règle de production qui spécifiera :

   si on a une expression composée de 2 fractions à additionner
   et si leur dénominateur est égal,

   alors l'expression obtenue en additionnant les deux numérateurs
   et en recopiant un dénominateur équivaut à la première expression.

   Nous représenterons cette règle de production sous la forme mathématique suivante :

   si $\frac{a}{b} + \frac{c}{d}$ et $b = d$

   alors $\frac{[a + c]}{b}$ où [a + c] signifie l'addition de a et c

   Une première étape dans l'élaboration de la base de connaissances consistera donc à relever toutes les règles de production utilisées pour résoudre un exercice faisant intervenir les opérations sur les fractions. Certaines seront identiques quelle que soit l'opération à effectuer, d'autres seront spécifiques à une opération.
   Ceci étant fait, notre base de connaissances contiendra les informations nécessaires pour résoudre des opérations sur les fractions. Mais nous désirons qu'elle contienne davantage. Nous voulons en effet que le système puisse suivre le raisonnement de l'élève et l'état actuel de la base de connaissances nous permet seulement de suivre l'élève modèle qui ne commet aucune erreur.

Nous allons donc rajouter des règles de production qui représenteront certaines étapes incorrectes effectuées par les étudiants. Par exemple, une des erreurs classiques de l'addition de deux fractions de même dénominateur sera représentée comme suit :

$$\text{si} \quad \frac{a}{b} + \frac{c}{d} \quad \text{et} \quad b = d$$

$$\text{alors} \quad \frac{[a + c]}{[b + d]}$$

où

[a + c] signifie l'addition de a et c

[b + d] signifie l'addition de b et d

Notre base de connaissances majorée de ces dernières règles répond maintenant bien à la définition donnée ci-dessus et fournira les connaissances nécessaires pour suivre le raisonnement d'un élève.

2. En ce qui concerne le système cognitif, il fonctionnera selon un algorithme très simple (Figure 3.4).
Comme on peut le constater, toute la difficulté de ce module d'inférence est reportée au niveau de la boîte noire que nous avons appelée "Tentative de détection de l'erreur"; nous la détaillerons ultérieurement.

3. Enfin, le dernier module de notre système expert (l'interface) sera assez simplifié. Dans le cadre de ce travail, en raison de différentes circonstances telles que le temps, la place mémoire, etc..., nous ne prendrons pas en considération toutes les caractéristiques de l'interface que j'ai citées ci-dessus.
C'est ainsi que, par exemple, la possibilité de poser des questions au système ne sera pas abordée; il en sera de même de la vérification syntaxique et sémantique des connaissances lors de leur acquisition.

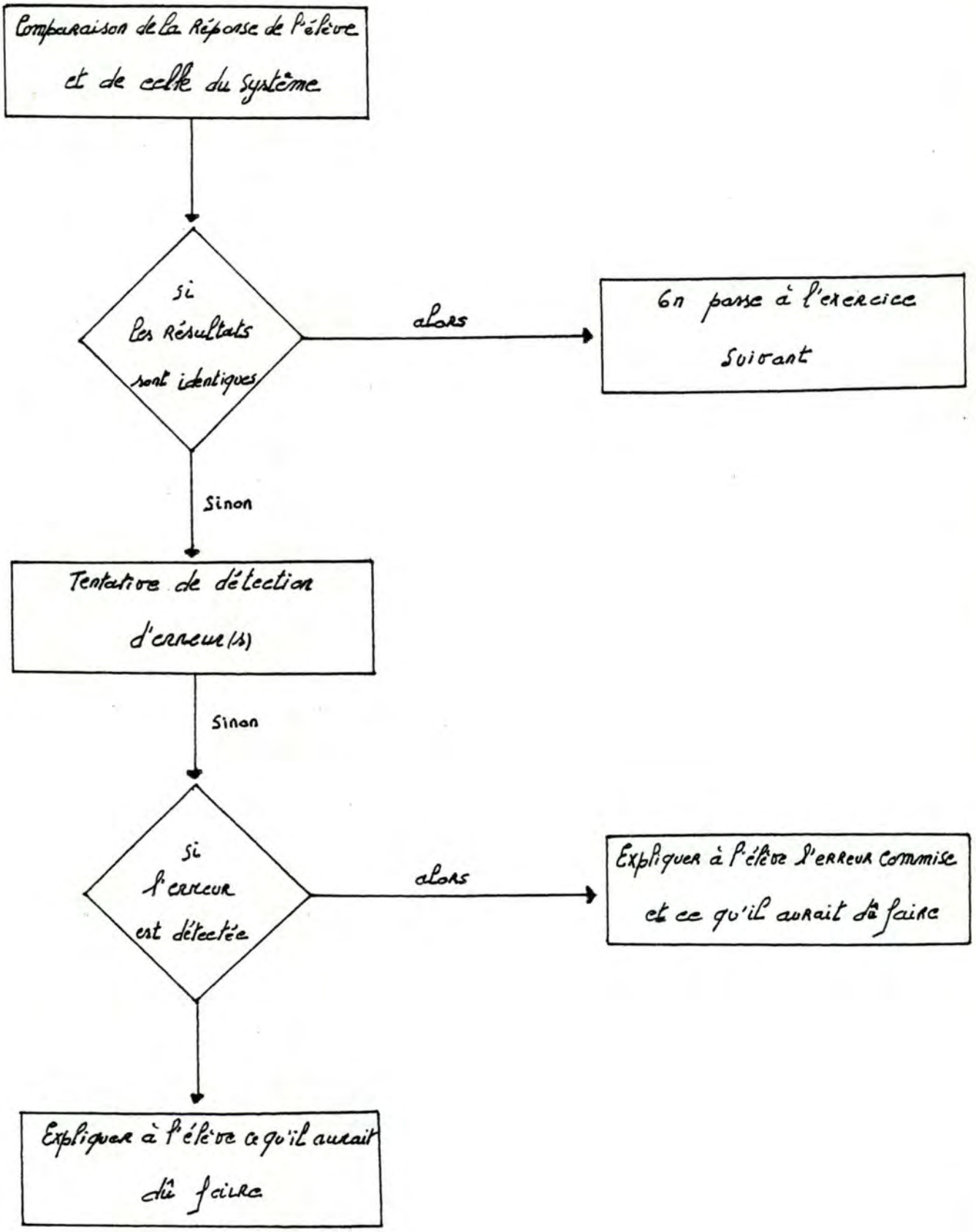

Comparaison de la Réponse de l'élève
et de celle du système
si
les résultats
sont identiques
alors
On passe à l'exercice
suivant
Sinon
Tentative de détection
d'erreur(s)
Sinon
si
l'erreur
est détectée
alors
Expliquer à l'élève l'erreur commise
et ce qu'il aurait dû faire
Expliquer à l'élève ce qu'il aurait
dû faire

Figure 3.4

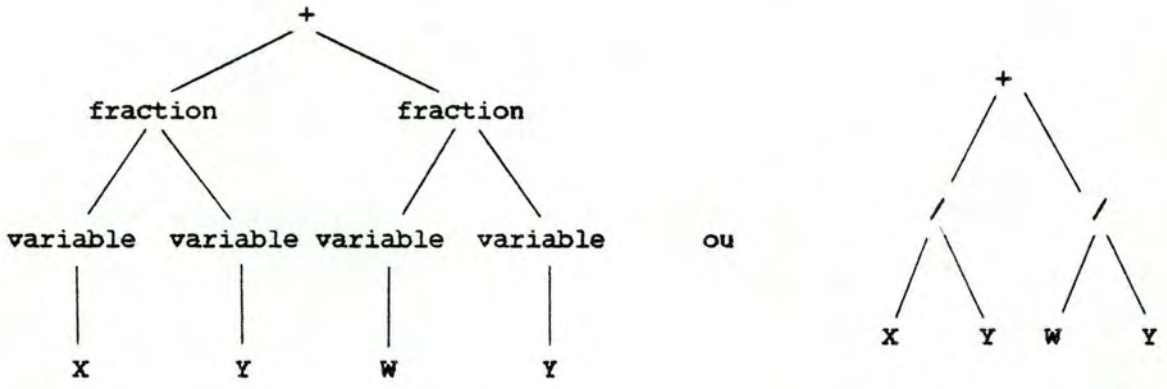
5.5. REALISATION

5.5.1. Représentation de connaissances au sein de la base de connaissances

La base de connaissances de notre système expert sera composée, comme nous l'avons dit, d'un ensemble de règles de production représentant la connaissance du domaine. Ces règles de production peuvent se représenter sous la forme d'un arbre syntaxique : par exemple, la règle

$$\frac{X}{Y} + \frac{W}{Y} \longrightarrow \frac{[X + W]}{Y}$$

peut se représenter sous la forme



Cet arbre syntaxique peut également se noter

R = (+ (/ X Y) (/ W Y))

D'autre part, les expressions que nous aurons à traiter peuvent également se noter sous cette forme.
Par exemple, l'expression 2/5 + 1/5 se notera

E = (+ (/ 2 5) (/ 1 5))

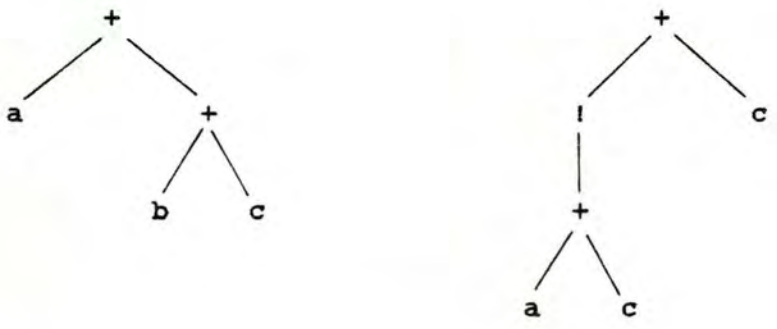
Comme on le constatera par la suite , cette représentation sous forme d'arbres syntaxiques a de nombreux avantages. Malheureusement, toutes nos règles de production ainsi que nos expressions ne peuvent se représenter sous la notation préfixée.

- C'est ainsi que, par exemple, l'utilisation des parenthèses est rendue insignifiante dans les arbres syntaxiques.

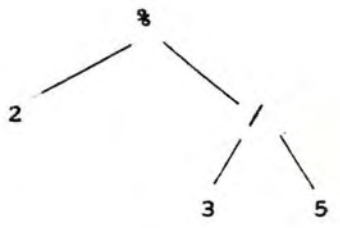
$$a + (b + c) \quad = \quad (a + b) + c \quad = \quad a + b + c$$

Or, celles-ci peuvent jouer un rôle primordial dans notre cas. C'est pourquoi, afin de réintroduire la signification des parenthèses, nous avons rajouté un opérateur d'arité 1 que nous noterons " ! ".
De cette manière, on peut distinguer l'expression

a + b + c de (a + b) + c



- Similairement, l'utilisation des arbres syntaxiques ne nous permet pas de représenter l'expression 2 3/5 (2 unités trois cinquièmes). Ici aussi, nous avons ajouté un nouvel opérateur d'arité 2 que nous noterons " % ".
L'expression 2 3/5 peut alors se représenter sous la forme



Le choix de ce formalisme de représentation des règles de production ainsi que la nature même d'une base de connaissances nous contraignent donc à réaliser les sous-routines décrites à la Figure 3.5.
En effet, comme les règles et les expressions seront représentées en notation préfixée, nous ne pouvons obliger l'éducateur ainsi que les élèves à employer cette notation. D'autre part, nous ne pouvons contraindre l'enseignant à fournir toutes les règles de production lors de chaque cession de travail.

Ces différentes procédures sont les suivantes :

- INREGLE : cette procédure enregistre une expression mathématique quelconque et la stocke dans un tableau.

  Si l'éducateur fournit, par exemple, la règle

  a /b + 2 / 8 <ret> ,

  elle sera stockée dans un tableau sous la forme :

  a / b + 2 / 8

- INFPRE : cette procédure traduit une expression en notation infixée en une expression en notation préfixée.

  La règle fournie par l'instituteur sera traduite en

  + / a b / 2 8

- STOREGLE : cette procédure stocke les différentes règles de production fournies par les éducateurs dans un immense tableau qui sera décrit dans la suite de cet exposé.

- PREINF : cette procédure, sur base d'un indice du tableau décrit ci-dessus, retraduit l'expression pointée en notation infixée.

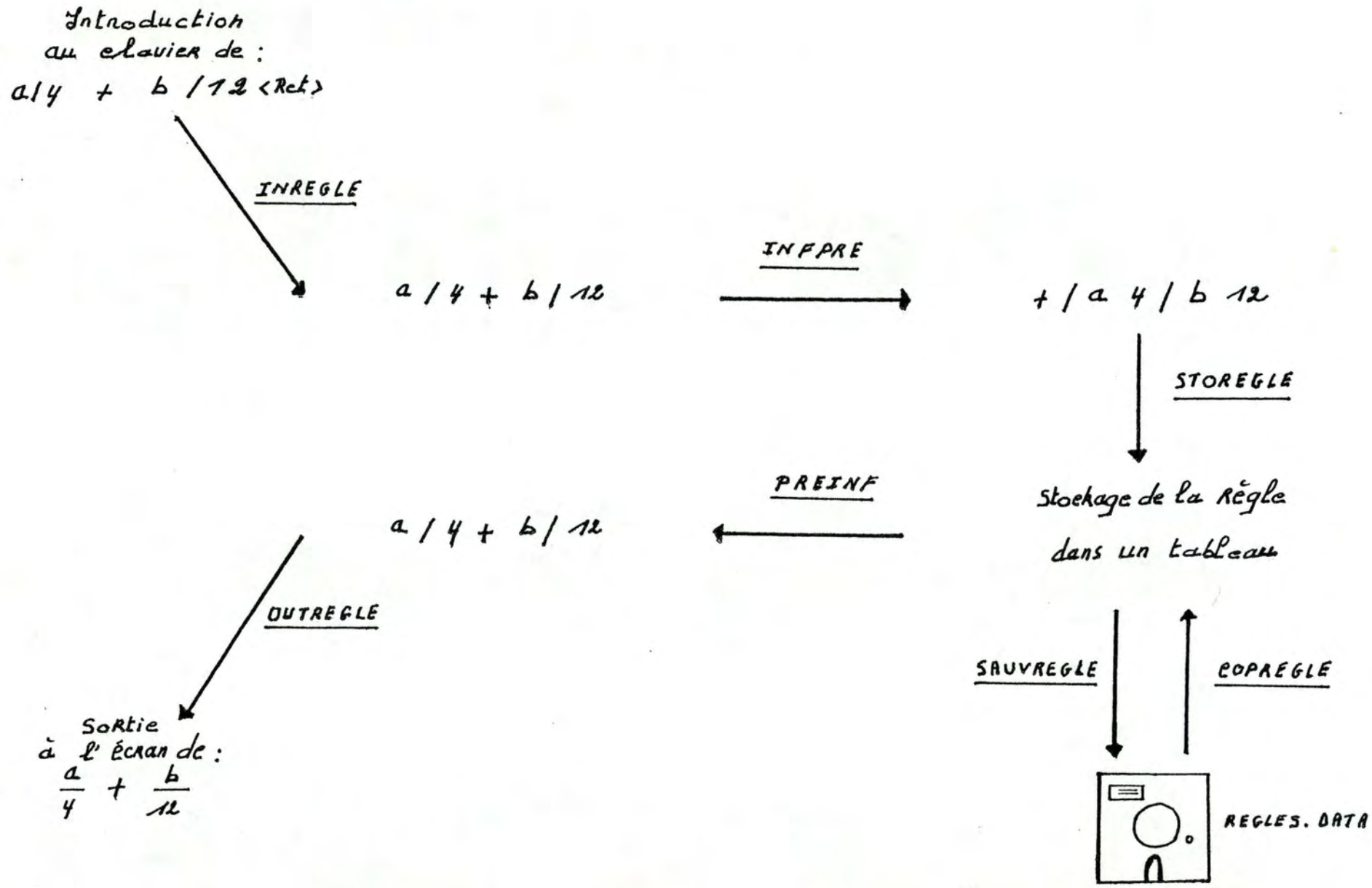
- OUTREGLE : cette procédure affiche à l'écran l'expression résultant de l'étape précédente. Notons cependant que, comme cette expression s'adressera aux élèves, nous avons pris soin de l'affichage.

L'exemple décrit sera affiché sous la forme :

$$\frac{a}{b} + \frac{2}{8}$$

- SAUVREGLE: cette procédure stocke le tableau cité lors de la description de la procédure "STOREGLE" dans un fichier (REGLES.DATA)

- COPREGLE : cette procédure est déclenchée au début d'une cession de travail. Elle copie le contenu du fichier "REGLES.DATA" en mémoire centrale.


Introduction
au clavier de :
a/4 + b/12 <Ret>
INREGLE
a / 4 + b / 12
INFPRE
+ / a 4 / b 12
STOREGLE
Stockage de la règle
dans un tableau
PREINF
a / 4 + b / 12
OUTREGLE
Sortie
à l'écran de :
a/4 + b/12
SAUVREGLE
COPREGLE
REGLES. DATA

5.5.2. Réalisation du système cognitif

1. Le mécanisme d'unification

Afin de bien cerner ce qui suit, résumons-nous en imageant la situation.

Supposons que l'on vous pose un problème auquel vous soyez tout à fait inconnu (par exemple, la traduction d'hiéroglyphes) et que, pour mener à bien ce travail, on vous fournisse un certain nombre de livres et de documents divers traitant de l'ère égyptienne. Votre démarche consistera à classer les différents documents, à y retrouver les alphabets utilisés dans l'Ancienne Egypte et à appliquer l'alphabet correct aux hiéroglyphes.
Les différentes étapes composant cette méthode de travail ne sont pas uniques et varieront d'un individu à l'autre; nous y reviendrons.... Cependant, l'étape consistant à appliquer les différents alphabets répertoriés aux hiéroglyphes jusqu'au moment où une traduction soit possible sera empruntée par chacun. De plus, si on calcule le temps nécessaire pour effectuer la traduction, ces tentatives d'unification d'un alphabet aux signes égyptiens sera déterminante.

Dans le cadre de ce mémoire, nous ne nous préoccupons pas de la traduction d'hiéroglyphes mais de la résolution d'un exercice traitant des fractions ou de la détection d'une erreur dans un raisonnement erroné. Pour mener à bien cette tâche, nous disposons d'une base de connaissances, composée, comme nous l'avons vu, de règles de production; celles-ci constituent la connaissance du domaine étudié et chacune d'elle représente une étape dans la démarche de raisonnement visant à résoudre le problème posé.

Si nous nous considérons dans les mêmes hypothèses que celles de la situation précédente (c'est à dire que nous ignorons tout du problème posé), notre démarche consistera à examiner chacune des règles et à tenter de l'appliquer au problème posé. Si une règle de la base de connaissances est applicable, nous exécuterons alors la transformation et nous recommencerons cette opération aussi longtemps qu'il le faudra.

L'opération consistant à examiner si une règle est applicable s'appelle l'unification. Examinons donc maintenant le détail de la réalisation de cette opération qui, rappelons-le, constitue un élément essentiel du temps de réponse.

Pour réaliser ce mécanisme, je me suis inspiré de l'algorithme général de Robinson ([ROBI, 1965], [ROBI, 1968], [ROBI, 1971]), algorithme que nous pouvons cependant simplifier fortement en raison de la spécificité du domaine étudié:

1. La règle à unifier à une expression sera toujours composée soit de variables, de constantes positives ou d'opérateurs (+, −, *, :, /, !, %).

2. L'expression à laquelle on tente d'unifier une règle comportera des constantes positives ou des opérateurs mais jamais de variables.

3. L'arité des opérateurs sera toujours inférieure ou égale à 2.

L'algorithme résultant de cette simplification est le suivant :

Soient rule = la règle à unifier,
expr = l'expression à unifier,
env = l'environnement, c'est à dire une table où l'on rangera le résultat des instantiations.

Considérons la règle et l'expression à unifier comme des tableaux et l'indice "i", l'indice du ième caractère de ces tableaux.

```
i = 1

si (le ième caractère de "rule") est une variable

    alors

        si (cette variable a déjà été instantiée)

            alors

                si (le ième caractère de "expr" correspond à
                    la valeur de l'instantiation)

                    alors    i = i + 1

                    sinon    l'unification est impossible;
                             on sort.

            sinon

                * On instantie la variable à la valeur du
                  ième caractère de "expr".
                * i = i + 1

si (le ième caractère de "rule") est une constante

    alors

        si (cette constante correspond à la valeur du ième
            caractère de "expr")

            alors     i = i + 1

            sinon     l'unification est impossible; on sort.

si (le ième caractère de "rule") est un opérateur

    alors

        si (l'opérateur de "rule" est différent de celui de
            "expr")

            alors

                l'unification est impossible; on sort.

            sinon

                on recommence l'unification pour chaque
                argument de cet opérateur.
```

Exemples :

1. règle : $\frac{A}{-} + \frac{C}{-}$ / B / 6

expression : $\frac{1}{-} + \frac{4}{-}$ / 2 / 6

résultats : l'unification est possible;
instantiation de A à 1,
de B à 2,
et de C à 4

2. règle : $\frac{A}{-} + \frac{C}{-}$ / B / 6

expression : $\frac{1}{-} - \frac{2}{-}$ / 2 / 6

résultats : l'unification est impossible;

3. règle : $\frac{A}{-} * \frac{C}{-}$ / B / B

expression : $\frac{1}{-} * \frac{3}{-}$ / 2 / 4

résultats : l'unification est impossible;

L'étape suivante de la réalisation de ce mécanisme correspond au choix de la structure de données utilisée pour représenter les règles. Ce choix sera déterminant quant à l'efficacité de l'unification et a donc nécessité de nombreuses optimisations.

La structure de données optimale utilise un tableau à trois colonnes pour représenter les règles de production :

- la première représentera le type du noeud de la règle ou de l'expression. (Il s'agira soit d'un opérateur, soit d'une variable, soit d'une constante).

  Ce type sera représenté par un entier strictement positif
  < 10 s'il s'agit d'un opérateur.
  = 20 s'il s'agit d'une variable.
  = 30 s'il s'agit d'une constante.

- la deuxième représentera, soit :

  1. le sous-arbre de gauche, si le noeud spécifié par la première colonne représentait un opérateur.
  2. une référence à une variable si le noeud specifié par la première colonne représentait une variable.
  3. la valeur de la constante si le noeud spécifié par la première colonne représentait une constante.

  Cette deuxième colonne contiendra toujours des entiers :

  1. si elle représente un argument, il s'agira d'un pointeur vers le sous-arbre de gauche.
  2. si elle représente une variable, il s'agira d'un entier compris entre 1 et 6 (puisque les seules variables que l'on puisse trouver sont notées a, b, c, d, e, f).
  3. si elle représente une valeur, il s'agira de la valeur elle-même.

- la troisième représentera, soit :

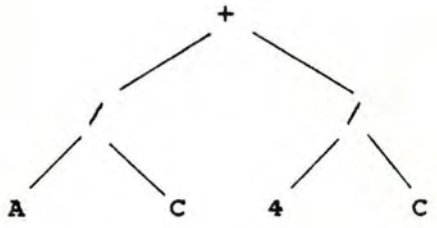
  1. le sous-arbre de droite si le noeud spécifié par la 1ère colonne représentait un opérateur et si l'arité de cet opérateur était supérieur à 1.
  2. rien dans les autres cas.

  Cette troisième colonne contiendra également toujours des entiers positifs :
  1. si elle représente un argument, il s'agira d'un pointeur vers le sous-arbre de gauche.
  2. si elle ne représente rien, la valeur "0" sera spécifiée.

Exemple :

L'expression $\frac{A}{C} + \frac{4}{C}$,

notée sous l'arbre syntaxique,



et représentée en notation préfixée,

+ / A C / 4 C

sera stockée sous la forme :

	COL 1	COL 2	COL 3
1	2 (+)	2	5
2	1 (/)	3	4
3	20 (variable)	1 (A)	0
4	20 (variable)	3 (C)	0
5	1 (/)	4	6
6	30 (constante)	4	0

Cette structure de données, optimale en termes d'unification peut-être, peut cependant nous sembler beaucoup moins intéressante quant à la place-mémoire dont elle est demanderesse. Il n'en est rien.... Examinons attentivement ce tableau à trois colonnes. Nous remarquerons que la ligne exprimant que "C est une variable" n'est reprise qu'une seule fois, alors qu'elle figure deux fois dans la notation préfixée.

Similairement, si nous désirons ajouter l'expression " - / A C / 4 C " dans ce tableau, nous obtiendrons le tableau :

	COL 1	COL 2	COL 3
1	2 (+)	2	5
2	1 (/)	3	4
3	20 (variable)	1 (A)	0
4	20 (variable)	3 (C)	0
5	1 (/)	4	6
6	30 (constante)	4	0
7	3 (-)	2	5

Trois entiers nous auront donc suffit pour représenter une expression supplémentaire.

Les routines découlant de ce mécanisme d'unification sont les suivantes :

1. Routine UNIF :

a. Spécifications concrètes

Cette routine sera capable d'affirmer si une unification est possible entre une règle et une expression. Si celle-ci est réalisable, elle fournira les valeurs d'instantiation.

b. Paramètres

Entrée :

- E1 : (1ère expression)
  Cette variable numérique contient l'adresse où est spécifiée la règle à unifier. Cette adresse représente en fait un indice du tableau que l'on a décrit ci-dessus.

- E2 : (2ième expression)
  Cette variable numérique contient l'adresse où est spécifiée l'expression à unifier. Cette adresse représente en fait un indice du tableau que l'on a décrit ci-dessus.

Sortie :

- ENV : (environnement)
  Ce tableau numérique contiendra les valeurs d'instantiation des différentes variables référencées dans les règles à unifier.
  Comme six variables nous suffisent, la longueur de ce tableau sera également de six; sa première valeur représentera éventuellement la valeur d'instantiation de la variable "A", la deuxième représentera éventuellement la valeur d'instantiation de la variable "B", ..., la dernière représentera éventuellement la valeur d'instantiation de la variable "F".

2. Routine STOREGLE :

a. Spécifications concrètes

Ce module a été décrit brièvement au paragraphe précédent mais, comme la structure de données utilisée pour représenter les règles de production n'avait pas encore été décrite, nous en avions postposé les spécifications précises.

Cette routine stockera les règles et les expressions dans un tableau tel que l'on vient de décrire. Cependant, comme on a pu le constater par des exemples, afin d'économiser la ressource rare que constitue la mémoire centrale, ce stockage tiendra compte du contenu du tableau afin d'éviter l'enregistrement d'informations identiques (ce traitement sera réalisé par la routine FILSTOCK).

b. Paramètres (OUTREGLE)

Entrée :

- PTEUR : (pointeur)
  Cette variable numérique contient l'indice du tableau à partir duquel la règle ou l'expression sera stockée.

- ENTREE : (tableau d'entrée)
  Ce tableau numérique contient la règle ou l'expression (en notation préfixée naturellement) à stocker dans la structure de données.

c. Paramètres (FILSTOCK)

Entrée :

- FILS : (fils)
  Cette variable contient, soit le sous-arbre de gauche, soit le sous-arbre de droite de l'expression ou de la règle à enregistrer dans la structure de données.

- I : (indice)
  Cette variable numérique représente l'indice courant du tableau contenant la description en notation préfixée de la règle ou de l'expression à enregistrer.

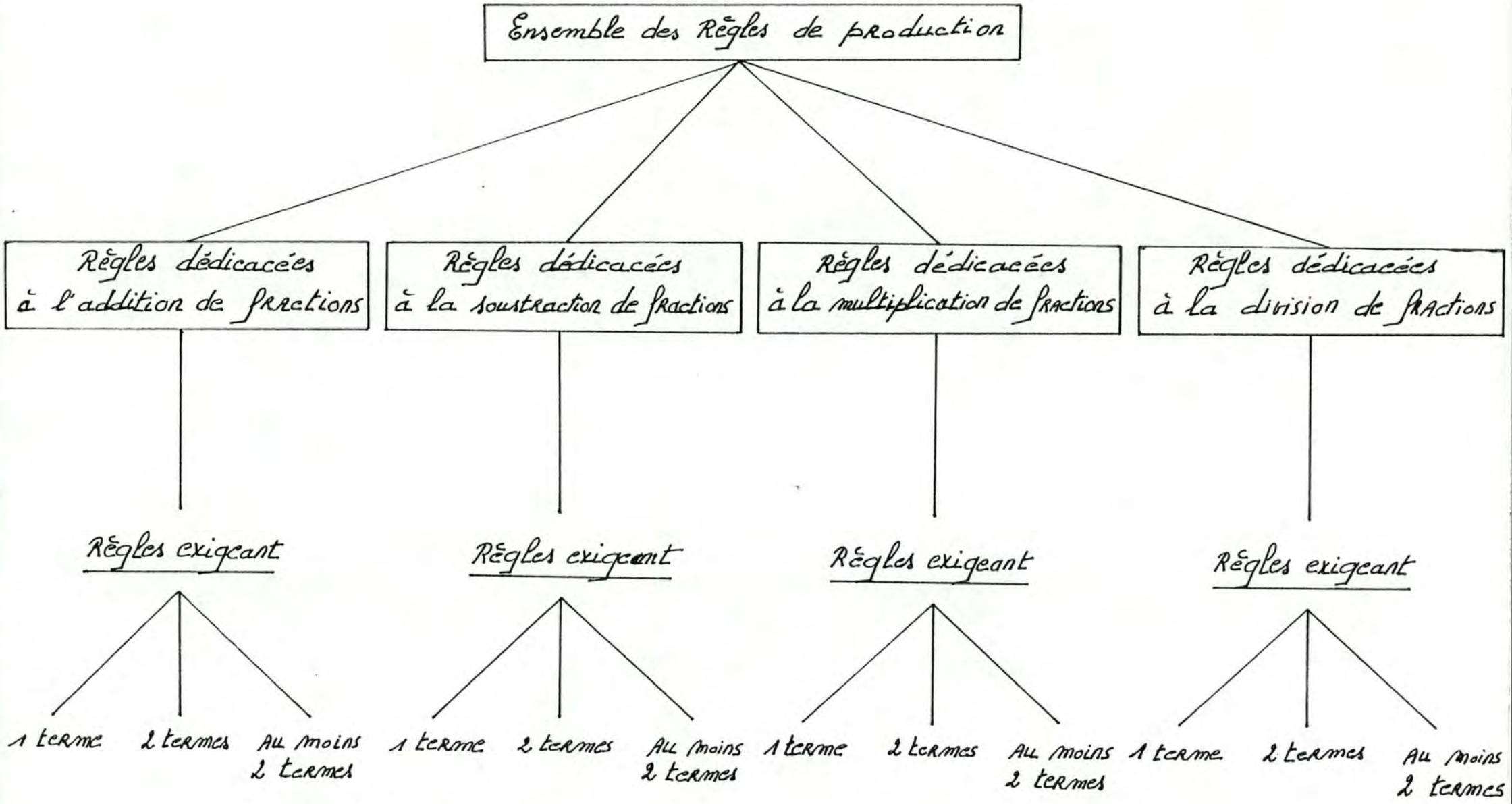
- CURTAB : (indice courant du tableau)
  Cette variable numérique contient l'indice courant du tableau dans lequel on stocke les différentes règles ou expressions. Il représente, à tout moment, l'indice du tableau où l'on doit écrire.

2. Le moteur d'inférence

Reprenons, si vous le voulez bien, notre exercice de la traduction d'hiéroglyphes. Nous y avons invoqué que la rapidité à résoudre le problème posé dépendait également de la méthode de travail utilisée. Celui qui a feuilleté les différents documents en y recherchant l'alphabet correct a pu avoir de la chance en l'identifiant assez rapidement. Mais, lors d'une deuxième traduction, son travail sera toujours aussi fastidieux.

Par contre, le résultat de celui qui a classé préalablement sa documentation s'est peut être fait attendre, mais ce dernier éprouvera certainement moins de difficultés pour les poblèmes à venir.

Nous avons réagi de la même manière avec les règles de production composant la base de connaissances. Comme nous avons à traiter des problèmes relevant des opérations sur les fractions, nous avons classé les règles selon le type d'opération à effectuer : certaines sont en effet uniquement dédicacées à un domaine particulier tandis que d'autres (les règles de simplification des fractions, par exemple) sont relatives à plusieurs problèmes.
Et, au sein de cette découpe, nous sommes encore parvenus à affiner les classes; en effet, il est tout à fait inutile de tenter l'unification d'une expression comptant deux termes à une règle en exigeant trois. La classification proposée à la Figure 3.6 nous permettra ainsi de minimiser les unifications à effectuer.


Ensemble des Règles de production
Règles dédicacées à l'addition de fractions
Règles dédicacées à la soustraction de fractions
Règles dédicacées à la multiplication de fractions
Règles dédicacées à la division de fractions
Règles exigeant
Règles exigeant
Règles exigeant
Règles exigeant
1 terme
2 termes
Au moins 2 termes
1 terme
2 termes
Au moins 2 termes
1 terme
2 termes
Au moins 2 termes
1 terme
2 termes
Au moins 2 termes

Ceci étant fait, nous pouvons maintenant envisagé la réalisation du moteur d'inférence proprement dit. Nous allons détaillé cette opération en relevant les différents modules nécessaires à sa réalisation, modules dont nous fournirons les spécifications concrètes à la fin de cette section.

Mais avant de nous lancer dans cette dernière phase relativement complexe, résumons-nous.
Etant donné le résultat d'un exercice traitant des opérations sur les fractions; résultat fourni par un élève, nous devons comparer cette réponse avec celle que nous avons calculée au moyen de règles contenues dans la base de connaissances. Si ces résultats sont différents, nous devons tenter de découvrir l'erreur de raisonnement opérée par l'étudiant et lui fournir ensuite le détail de la bonne démarche qu'il aurait dû suivre en lui faisant remarquer l'erreur commise.

La première étape consiste donc à résoudre le problème posé à l'élève afin de pouvoir porter un jugement sur son résultat. Comme nous l'avons rappelé, nous disposons, pour ce faire, d'un ensemble de règles de production représentant différentes étapes correctes de raisonnement ainsi que les spécifications du problème soumis à l'élève qui sont enregistrées dans un tableau caractérisant la structure de données que nous avons décrite précédemment (module "PROBL").

Pour mener à bien cette tâche, l'implémentation de cinq modules supplémentaires s'avère nécessaire :

- Le premier examinera si une règle est applicable à une expression. Vous nous direz que ce module n'est pas nouveau : nous avons déjà spécifié précédemment le module "UNIF" qui réalise ce mécanisme d'unification.
  Cependant, ce nouveau module que nous appelerons "PEUNIF" abrégé de "unification à une partie d'une expression", détectera, comme son nom l'indique, si une règle est applicable à une partie d'une expression.

  En effet, si nous considérons la règle et l'expression suivante :

$$\frac{A}{B} + \frac{C}{B} \qquad \frac{2}{6} + \frac{3}{6} \;,$$

  La routine "UNIF" nous fournira les valeurs d'instantiation des variables "A", "B", et "C".

Mais si l'expression à unifier à cette règle était la suivante

$$\frac{3}{6} - \frac{1}{6} + \frac{2}{6} \quad ,$$

cette même routine nous indiquerait que l'unification est impossible, alors qu'elle est réalisable sur le deuxième et troisième terme de cette expression.

La réalisation du module "PEUNIF" est donc entièrement fondée. Il utilisera cependant la routine "UNIF" pour tenter l'unification des sous-expressions avec la règle à unifier.

- Le deuxième module à réaliser dans le cadre de la réalisation du moteur d'inférence est dû, quant à lui, à un problème que nous avons ignoré jusqu'à présent.
En effet, si nous considérons la règle et l'expression suivante

$$\frac{A}{B} - \frac{C}{B} \qquad \frac{2}{6} - \frac{5}{6} \quad ,$$

l'unification est réalisable et le module "UNIF" nous fournira d'ailleurs les valeurs d'instantiation de la variable "A" à 2, de la variable "B" à 6 et de la variable "C" à 5.

Cependant, bien que cette unification soit correcte, le déclenchement de la partie "action" de cette règle de production ne peut être réalisée, sous peine de me fournir des résultats erronés.

En effet, nous avons omis de préciser une condition préalable au déclenchement du conséquent de la règle de production. Celle-ci spécifie que le dénominateur de la première fraction doit être supérieur ou égal à celui de la deuxième fraction, et donc que la valeur d'instantiation de la variable "A" doit être supérieure ou égale à celle de la variable "B" ... ce qui n'est pas le cas.

Ce nouveau module, que nous appellerons "VERIFCDT" vérifiera donc, dans le cas où l'unification est possible, si des conditions ne régissent pas éventuellement la règle de production unifiée et si les valeurs d'instantiation des variables les respectent.

- Les deux modules que nous examinerons maintenant sont relatifs au déclenchement du conséquent des règles de production.

  Si une règle est unifiable à une expression ou à une partie d'une expression, et si les conditions éventuelles régissant cette règle sont satisfaites, nous pouvons en effet appliquer la partie "action" de la règle à l'expression unifiée. Ce qui signifie que notre recherche de la solution au problème posé s'accroît d'une étape de raisonnement. Comme celle-ci doit être enregistrée afin que nous puissions expliquer à l'élève le raisonnement exact à tenir devant l'exercice à résoudre, ces nouvelles routines nous faciliteront la tâche.

  Comme dans le cas de la réalisation du mécanisme d'unification, deux modules sont nécessaires : le premier, que nous appellerons "DEDUIRE", sera déclenché si le nombre de termes de l'expression à unifier est identique à celui de la règle, tandis que le deuxième, que nous appellerons "PEDEDUIRE", sera activé dans le cas où une règle est unifiable à une partie d'une expression.

- Enfin, le dernier module, que nous appellerons "MOTEUR" constitue le coeur de ce système. Compte-tenu de la classification des règles de production (Figure 3.6), il agencera les différentes routines que nous venons de décrire jusqu'au moment où l'exercice sera résolu.

  Une question relative à la priorité accordée aux règles de production au sein d'une même classe se pose néanmoins; initialement, l'ordre dans lequel ces règles étaient considérées importait peu : si le système envisageait une mauvaise voie lors de sa recherche de la solution au problème posé (il le remarquait si aucune règle n'était plus applicable), il revenait alors à des étapes antérieures ignorant les démarches superflues; d'autre part, si le système bouclait (c'est à dire s'il rencontrait un cycle dans les étapes de raisonnement qu'il avait enregistrées), il pouvait détecter et éviter ce cycle en empruntant d'autres possibilités parmi les règles unifiables.
  Cependant, comme le temps de réponse de ce module était primordial et parce que le nombre de règles de production était peu élevé, nous avons pu, compte-tenu de la parfaite maîtrise du problème, affecter une priorité aux règles composant une même classe. Ceci nous a permis de gagner quelques dixièmes de secondes, voire même quelques secondes au temps de réponse de cette routine.

La deuxième étape de ce moteur d'inférence consiste à détecter l'erreur de raisonnement commise par l'élève si la réponse de ce dernier est différente de celle calculée par le système.

Les routines dérivant de cette deuxième étape seront identiques à celles décrites ci-dessus. La seule qui diffère toutefois est naturellement la routine que nous avions appelée "MOTEUR". En effet, celle-ci ne devra plus agencer les différents modules afin d'obtenir la solution d'un exercice mais pour découvrir l'erreur de raisonnement opérée par un étudiant.

Comme le nombre de règles de production à considérer dans ce cas est relativement élevé, puisque celles-ci ne sont pas exhaustives et parce que le nombre d'erreurs commises par l'étudiant est aléatoire, nous ne pourrons pas simplifier cette routine comme nous l'avons fait lors de la première étape. La seule possibilité pour soulager le travail du système et améliorer, par conséquent, le temps de réponse, consiste néanmoins à accorder une plausibilité aux différentes règles identifiant les types d'erreur. Ces plausibilités, caractérisant un élève particulier, seront d'ailleurs mises à jour régulièrement.
Cet artifice n'a pas la prétention d'éviter que le système n'emprunte des voies erronées ou qu'il ne perde du temps dans des cycles, mais nous espérons seulement qu'il lui permette d'améliorer son temps de réponse.

Avant de terminer la description de la réalisation du module "Examen de la réponse de l'élève" en détaillant le troisième et dernier composant de notre système expert, décrivons brièvement les routines à implémenter pour le moteur d'inférence.

1. Routine PROBL

a. Spécifications concrètes

Cette routine enregistrera au sein de la structure de données le problème soumis à l'élève. Celui-ci sera à l'origine du déclenchement du système expert.

b. Paramètres

Entrée :

- PTDEB : (pointeur de début)
  Cette variable numérique contient l'indice de la première ligne du tableau où insérer l'expression correspondant au problème posé à l'étudiant.

Sortie :

- ETAB : (élément du tableau)
  Cette variable numérique représente l'indice du tableau où insérer la prochaine expression.

2. Routine PEUNIF

a. Spécifications concrètes

Cette routine indiquera si une unification est possible entre une règle et une partie d'une expression.
Elle nous renseignera par exemple, si la règle et l'expression que l'on tente d'unifier sont les suivants :

```
A     C          2     1     3     5
- - -            - + - - - + -
B     B          8     8     8     8
```

b. Paramètres

Entrée :

- E1 : (1ère expression)
  Cette variable numérique contient l'adresse où est spécifiée la règle à unifier. Cette adresse représente en fait un indice du tableau caractérisant la structure de données.

- E2 : (2ième expression)
  Cette varaiable numèrique contient l'adresse où est spècifiée l'expression à unifier. Cette adresse représente en fait un indice du tableau caractèrisant la structure de donnèes.

Sortie :

- TROUVE : (unification possible ?)
  Cette variable booléenne indiquera si l'unification est possible ou non.

- I : (indice du terme)
  Cette variable numèrique indiquera le terme de l'expression à partir duquel l'unification est rèalisable avec la règle de production. (Dans notre exemple, "I" vaudra 2.

3. Routine VERIFCDT

a. Spècifications concrètes

Cette routine vèrifiera, dans le cas où l'unification est possible, si des conditions ne règissent pas èventuellement la règle de production unifiée et si les valeurs d'instantiation des variables les respectent.

b. Paramètres

Entrée :

- REAUNIF : (règle à unifier)
  Cette variable numèrique contient l'adresse où est spècifiée la règle à unifier. Cette adresse représente en fait un indice du tableau caractèrisant la structure de donnèes. En outre, cet indice identifie la règle.

- EXPAUNIF: (expression à unifier)
  Cette variable numèrique contient l'adresse où est spècifiée l'expression à unifier. Cette adresse représente en fait un indice du tableau caractèrisant la structure de donnèes.

Sortie :

- VERIF : (est-ce vèrifié ?)
  Cette variable booléenne indiquera si les conditions règissant la règle de production sont vèrifiées ou non.

4. Routine DEDUIRE

a. Spécifications concrètes

Cette routine appliquera le conséquent d'une règle de production à une expression comptant le même nombre de termes que la règle.

b. Paramètres

Entrée :

- RPTEUR : (pointeur vers la règle)
  Cette variable numérique contient l'adresse où est spécifiée la règle à unifier. Cette adresse représente en fait un indice du tableau caractérisant la structure de données.

- PTDEB : (pointeur de départ)
  Cette variable numérique représente l'adresse du tableau caractérisant la structure de données où insérer l'expression résultant de la déduction.

- FILS : (fils)
  Cette variable indique si nous devons traiter le sous-arbre de gauche ou le sous-arbre de droite de l'expression.

Sortie :

- ETAB : (élément du tableau)
  Cette variable numérique représente l'indice du tableau où insérer la prochaine expression.

5. Routine PEDEDUIRE

a. Spécifications concrètes

Cette routine appliquera le conséquent d'une règle de production à une expression comptant un nombre supérieur de termes que la règle.

b. Paramètres

Entrée :

- RPTEUR : (pointeur vers la règle)
  Cette variable numérique contient l'adresse où est spécifiée la règle à unifier. Cette adresse représente en fait un indice du tableau caractérisant la structure de données.

- PTDEB : (pointeur de départ)
  Cette variable numérique représente l'adresse du tableau caractérisant la structure de données où insérer l'expression résultant de la déduction.

- EXPTEUR : (pointeur vers l'expression)
  Cette variable numérique contient l'adresse où est spécifiée l'expression à unifier. Cette adresse représente en fait un indice du tableau caractérisant la structure de données.

- J : (indice)
  Cette variable numérique indique le terme de l'expression à partir duquel l'unification est réalisable avec la règle de production.

Sortie :

- ETAB : (élément du tableau)
  Cette variable numérique représente l'indice du tableau où insérer la prochaine expression.

6. Routine MOTEUR

a. Spécifications concrètes

Cette routine est le coeur du système. Elle constitue l'organe qui déclenchera toutes les routines que nous venons de décrire.

3. L'interface utilisateur

Dans le cadre de notre application, cet interface sera assez réduit (nous en avons déjà discuté à la section 5.4.2).
Les deux routines y afférent ont été décrites à la Figure 3.5. Nous n'y reviendrons donc pas.

6. CHOIX D'UN EXERCICE

6.1. Introduction

Comme nous l'avons souligné ci-dessus, notre étude des fractions se veut personnalisée. D'autre part, le choix d'un exercice proposé à l'élève doit tenir compte des aptitudes de celui-ci. Examinons donc la meilleure classification des exercices qui prendra en considération ces deux facteurs.

6.2. Classification des exercices

Une première possibilité consiste à répertorier un ensemble d'exercices traitant de tous les concepts englobant la notion de fractions, de les classer et de les numéroter en fonction de leur difficulté croissante. Chaque élève débutera alors l'étude de ce nouveau concept avec l'exercice 1 et sera supposé l'avoir complètement assimilé lorsqu'il sera parvenu à résoudre le dernier exercice. A tout moment, si les réponses proposées ne sont pas satisfaisantes, l'élève sera contraint de revoir les notions mal assimilées avant de poursuivre sa progression.

Une deuxième démarche consiste à relever les différentes notions nécesssaires à la compréhension des fractions. (Il s'agit notamment de la reconnaissance du dénominateur d'une fraction, de la reconnaissance du numérateur d'une fraction, de la comparaison de fractions, etc...). Comme nous pouvons le remarquer, l'assimilation de ces différentes notions se veut séquentielle : nous ne pouvons pas faire résoudre par l'élève des exercices traitant de la comparaison de fractions avant que celui-ci ne domine les concepts de dénominateur et de numérateur d'une fraction. Cependant, à la différence de la première possibilité proposée ci-avant, la démarche effectuée par les différents élèves pour maîtriser ces notions successives est personnelle : d'une part, certains domineront des concepts plus rapidement que d'autres; d'autre part, les difficultés rencontrées par les élèves sont propres à chacun d'eux. Une approche personnalisée se basant sur les caractéristiques propres de chaque enfant et non plus sur une séquence programmée d'exercices nous semble donc nécessaire.

Dans cette même optique, nous nous sommes attachés, avec l'aide de l'instituteur concerné, à détecter les difficultés que l'élève devra surmonter pour acquérir les différentes notions nécessaires à la compréhension des fractions.
L'acquisition de certains concepts nécessitant des représentations graphiques, la première difficulté envisagée nous sembla donc inhérente au moyen de représentation utilisé. Une deuxième difficulté consiste dans le nombre de parties équivalentes dessinées dans le support graphique (la fraction 1/2 ou 1/4 est plus facilement identifiable que la fraction 1/7 ou 1/12). Une troisième difficulté naît de la manière dont ces parties équivalentes sont représentées sur le support graphique. (Il y a en effet diverses possibilités de découper, par exemple, un rectangle en quatre parties équivalentes; certaines découpes semblent, pour l'enfant du moins, plus suggestives que d'autres). Il y a ensuite une quatrième difficulté qui résulte de la question posée à l'élève; dans le cas, par exemple, de la reconnaissance du dénominateur d'une fraction, trouver le nombre de parties équivalentes dessinées dans une figure et, d'autre part, représenter avec le symbolisme des fractions la signification d'une de ces parties

équivalentes semble bien un obstacle différent puisque la deuxième question implique l'aptitude à résoudre la première. L'existence de la cinquième difficulté est due, quant à elle, au cas particulier de certains exercices que nous voulions soumettre aux élèves. Il s'agit en fait d'exercices où l'on propose, par exemple, à l'enfant plusieurs représentations graphiques d'une même figure mais divisées en un nombre différent de parties équivalentes et où on lui demande de retrouver celle correspondant à telle fraction. Cette difficulté régira ici le nombre de représentations graphiques proposées à l'élève ainsi que s'il s'agit de figures identiques ou non. La sixième difficulté prendra en considération la nature des données. Elle s'appliquera surtout lors des opérations à effectuer sur les fractions. (En effet, additionner deux fractions de même dénominateur ou deux fractions dont l'une a son dénominateur multiple de l'autre ne nécessite pas le même raisonnement). Enfin, la dernière difficulté relevée tient compte de l'opération à effectuer sur les fractions.

6.3. Spécifications des difficultés

Nous avons relevé ci-dessus les sept difficultés suceptibles d'être rencontrées par les élèves. Nous allons maintenant les reprendre en les détaillant. La découpe présentée ici fut mise au point en collaboration avec un instituteur : nous n'affirmons pas qu'elle est unique, ni exhaustive (vous trouverez à l'appendice 3 un tableau reprenant l'ensemble de cette découpe).

- La difficulté 1 relative au moyen de représentation utilisé comportera neuf niveaux répartis par ordre croissant de difficulté, de la manière suivante : le carré, le carré sur pointe, le rectangle horizontal, le rectangle vertical, le cercle, le triangle équilatéral, l'hexagone, l'octogone et le segment.

- La difficulté 2 consistant dans le nombre de parties équivalentes dessinées dans le support graphique comportera huit niveaux de difficultés répartis comme suit : l'étude du demi, du quart, du tiers, du sixième, du huitième, du cinquième, du dixième et du douzième.

- La difficulté 3 née de la manière dont ces parties équivalentes sont représentées sur le support graphique comportera quatre niveaux qui concrétisent les différentes possibilités de découper une figure en parties équivalentes. Cette difficulté fut assez délicate à exprimer en raison de son interdépendance avec les deux premières. Vous trouverez à l'appendice 2 de plus amples explications concernant la spécificité de ce niveau de difficulté.

- La difficulté 4 résultant de la question posée à l'élève comportera onze niveaux répartis comme suit : trouver le nombre de parties équivalentes d'une figure; trouver le nombre de parties équivalentes d'une figure et le convertir en fraction; associer une fraction à une figure; associer n fractions à n figures; trouver le nombre de parties équivalentes coloriées d'une figure; trouver le nombre de parties équivalentes coloriées d'une figure et le convertir en fraction; représenter une fraction dans une surface; identifier deux fractions et les comparer; placer des fractions sur une droite graduée; trouver des fractions équivalentes à une autre; réaliser une opération arithmétique sur des fractions en simplifiant le résultat.

- La difficulté 5 comportera, quant à elle, sept niveaux selon que l'exercice proposé à l'élève comprend une, deux, trois ou quatre figures identiques ou bien deux, trois ou quatre figures différentes.

- La difficulté 6 prenant en considération la nature des données comportera quinze niveaux répartis de la manière suivante : une fraction inférieure à l'unité dont le numérateur vaut un; une fraction quelconque inférieure à l'unité; deux fractions de même dénominateur; deux fractions de même numérateur; deux fractions dont l'une a son dénominateur multiple de l'autre; une fraction équivalente à un naturel; une fraction supérieure à l'unité; une fraction simplifiable; deux fractions de même dénominateur inférieures à l'unité; un naturel et une fraction; deux naturels et une fraction; un naturel et deux fractions; deux naturels et deux fractions; un naturel et une fraction; une fraction et un naturel.

- La difficulté 7 tenant compte de l'opération à effectuer sur les fractions reprendra cinq niveaux : la simplification, l'addition, la soustraction, la multiplication et la division.

6.4. Spécifications des "canevas"

Examinons maintenant, d'une manière plus détaillée, les diverses étapes séquentielles que les élèves devront franchir. Celles-ci s'associent en fait à une parfaite compréhension des différents concepts correspondant à une maîtrise de ce que j'ai appelé des "canevas". Ces "canevas" correspondent aux buts à atteindre et seront "modulés" par les difficultés énumérées ci-dessus.

- Le canevas 1 intoduira le concept de "dénominateur d'une fraction". On proposera à l'élève une figure (disque, rectangle, polygône régulier, segment) divisé en un certain nombre de parties équivalentes et on lui posera diverses questions à ce sujet.

- Le canevas 2 introduira le concept de "dénominateur d'une fraction". On affichera plusieurs figures, et on fournira une ou plusieurs fractions que l'élève devra associer. (Ces figures seront identiques au début, quelconques ensuite, mais toutes seront divisées de manière différente).

- Le canevas 3 introduira le concept de "numérateur d'une fraction". On affichera une figure divisée en un certain nombre de parties équivalentes, et on demandera à l'élève de trouver ce nombre. Ensuite, on coloriera plusieurs de celles-ci et on posera à l'élève diverses questions à ce sujet.

- Le canevas 4 introduira également le concept de "numérateur d'une fraction". On affichera plusieurs figures divisées en un même nombre de parties équivalentes, mais le nombre de ces parties équivalentes coloriées étant différent. On fournira ensuite une ou plusieurs fractions que l'élève devra associer.

- Le canevas 5 terminera l'introduction du concept de "numérateur et de dénominateur d'une fraction". Il permettra à l'élève de représenter une fraction dans une surface.

- Le canevas 6 introduira le concept de "comparaison de fractions". On proposera deux surfaces identiques partagées en un certain nombre de parties équivalentes, certaines de ces parties étant hachurées. L'élève devra identifier les parties coloriées de ces figures et trouver si la première est plus petite, égale ou plus grande que la seconde.

- Le canevas 7 introduira également le concept de "comparaison de fractions". On proposera plusieurs fractions que l'élève devra placer sur une droite graduée.

- Le canevas 8 introduira le concept de "comparaison de fractions" et celui de "simplification de fractions". On proposera à l'élève diverses fractions que celui-ci devra simplifier au maximum.

- Le canevas 9 terminera l'introduction du concept de "comparaison de fractions". On proposera une fraction à l'élève qui devra trouver une ou plusieurs fractions lui étant équivalentes.

- Le canevas 10 introduira le concept "d'addition" et "de soustraction de fractions". On proposera à l'élève plusieurs figures : la première représentant la première fraction; la seconde représentant la fraction à additionner ou à soustraire; la dernière représentant l'opération effectuée. L'élève devra donc additionner ou soustraire les fractions et simplifier ensuite la réponse au maximum.

- Le canevas 11 terminera l'introduction du concept "d'addition" et de "soustraction de fractions". On proposera deux fractions que l'élève devra soit additionner, soit soustraire, et dont il devra simplifier le résultat.

- Le canevas 12 introduira le concept de "multiplication" et de "division de fractions". (cfr canevas 10)

- Le canevas 13 terminera l'introduction du concept de "multiplication" et de "division de fractions". (cfr canevas 11)

Il est bien évident que tous les niveaux de toutes les difficultés ne sont pas applicables à chacun des "canevas" (cfr appendice 4). C'est la raison pour laquelle nous avons dit que les "canevas" seraient "modulés" par les différentes difficultés. Ainsi, par exemple, en ce qui concerne le canevas 1, il sera module sur la difficulté 1 du niveau 1 à 9, sur la difficulté 2 du niveau 1 à 8, sur la difficulté 3 du niveau 1 à 4, sur la difficulté 5 du niveau 1 à 2, sur la difficulté 5 au niveau 1 et sur la difficulté 6 au niveau 1. Chaque élève débutera l'approche de ce canevas avec tous les niveaux des difficultés énumérées valant un. Son but est de prouver sa bonne compréhension de la notion incorporée dans le canevas en réussissant à surmonter toutes les difficultés, amenant ainsi les niveaux de celles-ci à leur maximum. Pour ce faire, il réalisera correctement un premier exercice dont les caractéristiques sont issues des niveaux des différentes difficultés. Ensuite, compte tenu de son "historique, on lui soumettra un autre exercice, caractérisé par un niveau de difficulté différent du précédent, et, ainsi de suite jusqu'au moment où le but sera atteint.

Remarquons que deux exercices successifs se distincteront par un des niveaux d'une des difficultès diffèrent; ce qui ne veut pas dire que la progression de l'élève nècessitera le passage par chacun des niveaux de difficultès. Un étudiant qui a prouvè sa maîtrise d'une des difficultès peut passer très rapidement au niveau maximum de celle-ci et, par contre, avancer progressivement dans une difficultè encore mal assimilèe. Notons ègalement que cette approche permet une dètection rapide et aisèe des concepts mal compris. Si le passage du niveau "i" d'une difficultè au niveau "i + 2" n'est pas surmontè par l'èlève, la cause ne peut en être qu'une progression trop rapide au sein de celle-ci. On reviendra donc en arrière et on reprendra une èvolution plus lente.

La dèmarche adoptèe variera donc très fort d'un èlève à l'autre. En route vers son objectif de comprèhension des fractions, chaque étudiant passera donc par certaines ètapes identiques pour tous, mais pour atteindre ces points de repère, comme aucun tracè n'est prèalablement établi, chacun avancera à son propre rythme et par le chemin qui lui convient le mieux. Afin de bien percevoir cette dèmarche, examinons la Figure 3.7; les axes repràsentent deux difficultès inhèrentes à l'étude des fractions (dans le cas rèel, nous en aurons sept) et les "o" rerèsentent les diffèrents canevas (dans le cas rèel, nous en avons retenu treize). Chaque èlève ètant supposè n'avoir aucune notion des fractions dèbutera donc son étude au point 1 et sera supposè dominer la matière lorsqu'il atteindra le point 2. Mais entre ces deux extrêmes, comme l'illustrent les lignes en pointillès, une multitude de chemins sont possibles.

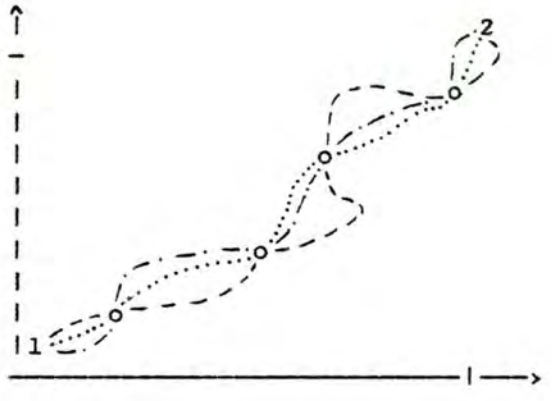


Figure 3.7

C H A P I T R E 4

IMPLEMENTATION DES DIFFERENTS MODULES

Comme nous l'avons déjà souligné, nous n'étudierons présentement que l'implémentation des modules développés lors du chapitre précédent. Pour chacun d'eux, une étude détaillée sera exposée et le lecteur sera invité à se reporter aux annexes référencées s'il désire examiner le code de ces différents modules.
Nous examinerons ensuite les différentes techniques que nous propose le micro-ordinateur afin de rassembler ces différentes routines en un programme complet et efficace.

1. INTRODUCTION

Le matériel que l'école primaire de Don Bosco nous propose se compose actuellement d'un apple II E et d'un moniteur noir et blanc. Il est toutefois possible de disposer des quatre apple II (et de leur moniteur noir et blanc) qui appartiennent au cycle secondaire.

Cette configuration nous offre plusieurs possibiltés quant au langage de programmation à utiliser; le Basic, le Pascal, le Logo et le langage Prolog sont en effet disponibles sur ce type de matériel. Le langage Prolog a retenu notre attention pour ses facilités à implémenter un système expert. Le langage Logo nous a attirés par ses potentialités graphiques. L'utilisation du langage Pascal et du langage Basic ont suscité notre intérêt en raison de leur caractère standard.

Cependant, le Logo et le Prolog étant très exigeant en place-mémoire et comme notre projet est relativement conséquent, notre choix définitif s'est porté sur le langage Pascal.

2. MODULES "AFFICHAGE, PARTAGE ET COLORIAGE DE FIGURES"

La réalisation de l'appui graphique auquel sont dédicacés ces trois modules nécessitant une précision relativement importante, nous travaillerons en mode graphique. Moyennant l'utilisation d'une librairie (SYSTEM.LIBRARY), le language Pascal peut accéder à un certain nombre de procédures et de fonctions lui permettant de travailler en coordonnées cartésiennes sur un écran dont la définition est de 280 x 192 points. Ces routines contenues dans la librairie permettent également de travailler avec différentes couleurs; malheureusement, l'école en collaboration avec laquelle nous travaillons ne disposant pas d'écran couleur, nous nous contenterons des deux couleurs fondamentales.

Mais le grand avantage que peut nous apporter l'utilisation de cette librairie coûte très cher : son stockage en mémoire centrale. En effet, bien que nous n'exploiterons qu'une partie des facilités offertes par la librairie, son stockage intégral est absolument nécessaire. Et comme celui-ci est conséquent, nous prîmes donc la décision de réécrire les quelques routines nécessaires en assembleur (puisque Apple, naturellement, nous interdit l'accès au code de sa librairie). Ce travail fut assez long et prit plus d'un mois mais, que de déception après cette tâche.... En effet, l'élaboration des autres modules du logiciel s'affinant, nous avons constaté que l'utilisation de la librairie graphique ne serait pas seulement limitée à ces trois modules et que cette extension impliquerait

l'emploi de nouvelles possibilités, de telle sorte qu'une grande partie de la librairie serait nécessaire.
C'est pourquoi, nous avons interrompu l'implémentation de ces modules en assembleur et avons accepté le prix à payer pour l'utilisation de la librairie graphique. (Vous trouverez à l'annexe 7 l'implémentation en assembleur de la routine qui permet de tracer une ligne d'un point à un autre).

Un autre problème non trivial se devait également d'être traité. Il est dû au fait que les points composant l'écran en haute résolution n'ont pas leur longueur et leur largeur équivalentes. Nous ne rentrerons pas dans le détail de la raison de cette constatation mais, toujours est-il, que le tracé d'un carré par exemple me fournira un rectangle, et que le tracé correct d'un carré sur pointe exigera que la somme des angles soit égale à 360 degrés mais que les angles opposés soient égaux 2 à 2 et différents de 90 degrés.
Nous ne pouvons vous fournir les coefficients exacts de correction à porter à ces différents cas puisqu'ils varient d'un type d'écran à un autre.

L'utilisation des trois modules que nous décrivons est transparente à ces problèmes, mais cette remarque devait être signalée pour une bonne compréhension du code Pascal de ces routines dont vous trouverez le détail à l'annexe 1.

3. MODULE "GENERATION DE DONNEES"

Afin d'obtenir une entière satisfaction sur la qualité des données, celles-ci devront être générées par une fonction aléatoire.
Moyennant l'utilisation d'une librairie (différente de celle utilisée pour la réalisation de l'appui graphique), le langage Pascal peut accéder à une fonction qui assurera l'uniformité des valeurs générées par rapport à un intervalle fixé. Cependant, comme dans le cas des trois modules précédant, l'utilisation de cette fonction nous coûtera le stockage intégral de la librairie en mémoire centrale.

Nous avons donc envisagé de construire notre propre fonction aléatoire. La difficulté ne résidait pas dans la génération proprement dite mais il nous fallait trouver un nombre semence aléatoire duquel on pouvait déduire les autres et le seul que l'on a pu trouver est fonction du temps mis par l'étudiant pour fournir son nom au système.

Comme celui-ci ne nous satisfaisait pas pleinement et comme la librairie contenant la fonction aléatoire était relativement peu exigeante en termes de place-mémoire, nous avons en définitive choisi la solution la plus sûre et avons adopté la fonction fournie par le système. (Vous trouverez à l'annexe 2 l'implémentation de ce module).

4. MODULE "EXAMEN DE LA REPONSE DE L'ELEVE"

L'implémentation des différentes routines composant ce module fut rendue complexe par la perpétuelle obsession d'obtenir un temps de réponse idéal (c'est à dire de quelques secondes).

1. Les routines se rapportant au stockage des connaissances au sein de la base des connaissances ont toutefois échappé à cette course contre le temps. Vous les trouverez à l'annexe 3.

2. Les routines afférent au moteur d'inférence ont subi, quant à elles, de multiples optimisations.

   Vous trouverez à l'annexe 4 plusieurs procédures d'unification résultant de différentes structures de données ainsi que plusieurs versions d'optimisation progressive dédicacées à la structure de données jugée optimale. Ces différentes versions ne constituent pas l'ensemble de nos recherches puisque nous avons expérimenté trois structures de données et que la version actuelle du module d'unification a transité par plus de dix étapes d'optimisation. Cette dernière réalise cent cinquante fois l'unification de la règle et de l'expression de la Figure 4.1 en 7 secondes, alors que la première fonction que nous avions réalisée en mettait plus de 20.

```
A       C           1       2
-   +   -           -   +   -
B       B           4       4
```

Figure 4.1

Examinons brièvement la dernière version de l'implémentation du mécanisme d'unification. Les deux premières lignes correspondent à des options du compilateur.
La première permet l'utilisation de l'instruction "GOTO" : cette instruction n'est pas proscrite du langage Pascal mais des approches plus structurées telles que le "FOR", le "WHILE" ou le "REPEAT" sont plus appropriées. Pour notre part, nous avons utilisé cette instruction exceptionnelle pour des raisons évidentes d'optimisation.
La deuxième option, quant à elle, permet au compilateur de ne plus produire du code vérifiant les références aux tableaux et aux strings. En effet, l'option par défaut oblige le compilateur à produire du code qui vérifiera, par exemple, que tout accès à un tableau via son indice est compris entre la borne inférieure et supérieure. Cette option est naturellement dangereuse et ne peut être utilisée que si le programme est entièrement correct et si des conditions d'optimisation l'exigent.

L'annexe 5 reprend les autres routines composant le moteur d'inférence. Ces dernières sont également très optimisées et exploitent abondamment la récursivité du langage Pascal; ce qui peut nous amener à des constatations étranges si l'on tente de dresser l'architecture des différents modules à partir du listing : on peut ainsi remarquer, par exemple, que la routine "VERIFCDT" utilise la procédure "PEUNIF" qui utilise elle-même la procédure "VERIFCDT" ...

3. Les routines constituant l'interface-utilitsateur que vous trouverez à l'annexe 3 ne comportent rien de particulier, si ce n'est le module "OUTREGLE".
Nous désirons en effet y soigner la présentation des fractions à l'écran; or, la composition du clavier d'un Apple II ne nous permet pas de représenter la fraction

$$\frac{1}{8}\frac{4}{}\ ,$$

mais seulement la fraction

$$\frac{1}{8}\frac{4}{}\ ,$$

ce qui ne nous satisfait pas.
Nous avons donc été contraint de redéfinir un nouveau caractère qui nous permette de réaliser le but que nous nous étions fixé.

5. MODULE "CHOIX D'UN EXERCICE"

Etant donné la structure de données adoptée, l'implémentation de ce module n'a causé aucune difficulté.

6. L'IMPLEMENTATION D'UN PROGRAMME COMPLEXE

L'implémentation de notre projet sur un micro-ordinateur nous a amenés à réagir face à deux inconvénients que celui-ci nous impose, à savoir la capacité mémoire limitée et la faible vitesse d'exécution qui en résulte.

Comme nous avons éclaté en plusieurs annexes les différentes routines résultant de l'analyse fonctionnelle et organique afin d'aider le lecteur dans sa compréhension du code Pascal généré, nous avons repris à l'annexe 6 le programme qui permet au système de résoudre un exercice sur les fractions. Ce-dernier concrétise en effet les différentes techniques fournies par le système Pascal de l'Apple II pour pallier les deux inconvénients précités.

Nous examinerons d'abord les techniques proposées pour surmonter la place mémoire limitée; nous étudierons ensuite les techniques relatives à l'amélioration de la vitesse d'exécution. Nous nous limiterons cependant aux techniques utilisées dans le cadre de notre projet.

6.1. Remèdes proposés pour compenser la place-mémoire limitée

1. Gestion de gros programmes-sources

Lors de la phase d'édition d'un programme, l'éditeur de texte et le programme sont présents en même temps en mémoire centrale. Dès lors, il y a souvent trop peu de place pour implémenter un programme en un seul fichier. Nous devons donc diviser celui-ci en un ensemble de petits fichiers que nous créons et modifions séparément. Le P-Système offre deux possibilités pour combiner plusieurs fichiers sources en un seul programme :

a. Le fichier "INCLUDE"

Le fichier "INCLUDE" permet de combiner des parties de programme-source qui ont été créées sous la forme de fichiers séparés, en un seul programme, sans en changer la signification.

La directive "INCLUDE", notée (*$I nom-du-fichier *), indique au compilateur le fichier à inclure ainsi que l'endroit où cette inclusion doit s'opérer dans le programme principal.

Un exemple d'utilisation de cette directive est présenté dans le programme "SE" de l'annexe 6. Nous y avons en effet séparé le fichier "GENERER" contenant les différentes routines nécessaires à la génération de données que nous devons inclure dans le programme principal.

b. La compilation séparée par l'intermédiaire des "UNITs"

Une "UNIT" est un ensemble de procédures, de fonctions et de déclarations Pascal qui forment une section indépendante de programme. Un dispositif Pascal UCSD permet d'ailleurs de réaliser la compilation séparée des "UNITs".

Le résultat de la compilation d'une "UNIT" est une librairie. Celle-ci peut être utilisée directement ou être incorporée à une autre librairie comme par exemple, la librairie SYSTEM.LIBRARY.

Il y a deux sortes de "UNITs" : les "REGULAR UNITs" et les "INTRINSIC UNITs". Lorsqu'un programme utilise une "REGULAR UNIT", le code de la "UNIT" est inséré dans le code du programme hôte, ou programme principal, par un éditeur de liens (le "LINKER"). Lorsqu'un programme utilise une "INTRINSIC UNIT", le code de la "UNIT" reste dans le fichier librairie et n'est pas intégré au programme-hôte. Le code de la "UNIT" est chargé en mémoire centrale au moment de l'exécution du programme principal et l'édition des liens n'est donc plus nécessaire. Ce genre de "UNIT" est, par conséquent, particulièrement intéressante lorsque beaucoup de programmes l'utilisent puisqu'elle permet de réduire la taille du fichier code du programme-hôte.

Afin de pouvoir utiliser une "UNIT", un programme doit contenir la directive "USES" suivie du nom de la "UNIT".

Le programme de l'annexe 6 que nous avons pris à titre d'exemple utilise cinq librairies ou "UNITs" : les librairies "APPELSTUFF" et "TURTLEGRAPHICS" dont nous avons précédemment expliqué le contenu correspondent à des "INTRINSIC UNITs" tandis que les trois suivantes sont des "REGULAR UNITs". (Vous trouverez également dans cette annexe 6 l'explicitation de ces-dernières).

Enfin, une dernière remarque concernant la gestion de gros programmes-sources doit encore être notée. Elle ne correspond pas à un remède proposé par le système mais plutôt à un "truc" pour surmonter les limites imposées.

En effet, le compilateur ne nous permet pas de créer des procédures ou des fonctions dont la taille du code-object dépasse 1 200 bytes. La création de telles routines peut dénoter une mauvaise programmation ou une mauvaise analyse organique, mais ce n'est pas toujours le cas. Pour vaincre cette limite, nous nous sommes donc proposés de scinder en deux la procédure trop importante en attribuant un nom aléatoire à une de ces parties.

La procédure "P2UNIF" de notre programme, associée à la procédure "PEUNIF", ne doit donc son nom qu'à ce besoin et ne doit pas influencer le lecteur quant à la signification de son contenu.

2. Compilation de gros programmes-sources

Le problème majeur de la compilation de gros programmes, et en particulier de grosses "UNITs", consiste en la taille de la mémoire disponible pour recevoir le compilateur et la table des symboles.

Si, durant la compilation, l'espace-mémoire n'est pas suffisant, le compilateur s'arrête, mentionnant une erreur de débordement de pile. La solution consiste alors à utiliser les options de recouvrement qui permettent de libérer de la place pour la table des symboles. Ces options de SWAPPING, notées (*$S+*) et (*$S++*), permettent respectivement de libérer 5 000 et 6 500 mots supplémentaires.
Ces options n'ont aucune conséquence sur le code généré par le compilateur; seule, une compilation plus lente témoigne de l'utilisation de celles-ci.

3. Exécution de gros programmes

L'espace-mémoire est également une ressource critique pour l'exécution de programmes importants. Pour remédier à ce problème, nous pouvons agir en mentionnant de nouvelles options.

En effet, lors de l'exécution de programmes utilisant des librairies ou des "UNITs", le code de toutes ces extensiona se trouve habituellement en mémoire centrale, ainsi que le code du programme principal. Mais, si la quantité de ce programme exécutable est trop élevée, deux options nous permettent de gérer le code à conserver en mémoire centrale et celui à amener en mémoire sur une demande implicite.

Ces deux facilités offertes par le système sont l'option "NOLOAD", notée (*$N+*), qui chargera en mémoire le code d'une "UNIT" uniquement si une des routines la composant est activée et l'option "RESIDENT", notée (*R nom-d'une-procédure *).

6.2. Remèdes proposès en vue d'amèliorer la vitesse d'exècution

La vitesse d'exècution relativement faible est liée à la nature même d'un micro-ordinateur : un Apple II n'est pas comparable à un Vax ou à un Dec 20/60. Or, comme nous l'avons dèjà souligné, un temps de rèponse satisfaisant constitue la condition sine qua non de l'utilisation de notre logiciel. L'implèmentation de nombreuses routines ont donc suscitè de multiples optimisations en vue de garantir un temps de rèponse convenable.

Dans cette optique, l'option "NO RANGE CHECK" du compilateur, notèe (*$R-*), nous a permis d'économiser de prècieuses secondes pendant lesquelles le code produit par le compilateur vèrifait la syntaxe de tous les accès aux tableaux. Nous pouvons en effet parler de secondes et non de dizièmes de secondes vu l'intense utilisation de tableaux que prèconise le logiciel.

Malheureusement, ce sont les techniques permettant de remèdier à la limitation de la place mèmoire qui annulent l'effet bènèfique de ces dernières options. L'utilisation du swapping par exemple, tellement utile pour pallier l'inconvènient prècèdent , coûte très cher à la vitesse d'exècution du programme.

L'utilisation de ces techniques, très utiles nèanmoins, doit donc être rèflèchie puisqu'aucun abus n'est tolèrè.

CONCLUSION

Après un résumé de la situation de notre travail et de son contexte, nous examinerons successivement les faiblesses et les points forts du didacticiel. Le dernier point sera consacré aux extensions envisageables et ouvrira la voie à des perspectives nouvelles en matière d'Enseignement Assisté par Ordinateur.

Réalisation du travail

Pour réaliser ce projet, nous avons opté pour la collaboration enseignant-informaticien, l'un apportant ses connaissances en pédagogie, l'autre en informatique. Cette formule de travail, même si elle requiert un investissement de temps important, nous semble cependant tout à fait adéquate et particulièrement enrichissante pour les deux parties : elle permet de mettre à profit les compétences de chacun, et oblige en outre à formuler complètement et clairement ses idées.

Rappelons que l'objectif de départ de ce projet consistait en la réalisation d'un didactiel visant à améliorer l'étude et la compréension de la notion de fractions dans le cadre de l'enseignement primaire.

Pour ce faire, nous avons relevé les différentes étapes séquentielles qui doivent être assimilées par l'élève. Nous nous sommes ensuite attachés à cerner les difficultés que les étudiants devront surmonter pour acquérir les notions précitées. Ces deux démarches combinées nous permettront de suivre l'évolution de l'élève à travers un tableau à trois dimensions : la première représente la notion courante assimilée, la deuxième concrétise la difficulté suggérée et la troisième symbolise le niveau de cette difficulté soumis à l'étudiant. L'élève désireux d'étudier la notion de fraction débutera donc l'étude de ce nouveau concept au premier niveau de ces trois dimensions et sera censé maîtriser la matière lorsqu'il atteindra le niveau maximum de chacun des trois axes.

Cette approche a facilité la réalisation de deux problèmes que nous nous étions proposés de traiter. Le premier consistait à permettre une approche personnalisée de l'étude d'un nouveau concept. Or, la démarche que nous avons adoptée peut satisfaire aisément ce désir puisque le chemin emprunté par chaque élève sera calqué sur ses antécédents.

D'autre part, cette même démarche se prête également à l'identification de l'origine des erreurs éventuelles commises par les étudiants étant donné que deux exercices successifs se distingueront uniquement par la variation d'une dimension du tableau. En outre, dans cette même optique, grâce à une typologie des erreurs les plus fréquentes, typologie que nous avons élaborée à partir d'exercices fournis à des élèves, nous avons pu envisager un processus d'interprétation et de remédiation des erreurs.

Nous avons mené ce projet complètement jusqu'à l'analyse fonctionnelle. Nous avons également terminé la conception de la partie traitant de l'étude et de la compréhension des fractions. L'implémentation de celle-ci reste cependant à terminer : il s'agit d'un simple codage de quelques modules et d'une intégration avec ceux existants puisque l'analyse résultant de cette étude est complète et détaillée.

Une des grandes satisfactions que ce travail nous a procurées est d'avoir convaincu des enseignants et des parents de l'intérêt que peut apporter l'utilisation de l'ordinateur au sein des écoles. Grâce à plusieurs expérimentations de diverses phases du logiciel, des personnes jusqu'alors indifférentes à l'informatique, ont en effet marqué le désir d'en savoir plus. Cet intérêt nous ayant touché, nous nous sommes d'ailleurs engagés à terminer ce didactiel.

Points faibles

Nous avons relevé, en collaboration avec des instituteurs, un point faible important à ce logiciel.

Il se situe au niveau de la remédiation. La conception de cette phase suppose en effet que la présentation de la bonne démarche à suivre complétée éventuellement de la description de(s) erreur(s) commise(s), permettra à l'élève qui a fourni des résultats erronés de comprendre le raisonnement incorrect qu'il a tenu.

Cependant, cette approche ne sera pas toujours suffisante et la possibilité de consulter un texte explicatif détaillé constitue un apport considérable.

Toutefois, la potentialité de poser des questions au système résolverait ce problème de manière plus souple. Mais, à la différence de la première solution proposée, la réalisation de cette dernière implique un investissement de temps et de travail très important.

Points forts

La force de ce didactiel peut se résumer en trois points fondamentaux : le premier concerne l'extensibilité et la flexibilité du logiciel; le deuxième réside dans son indépendance du niveau de l'enseignement visé et du sujet choisi; le troisième est dû à l'intérêt que nous avons porté à traiter la remédiation et le suivi des élèves.

La démarche que nous avons adoptée permet d'étendre aisément les trois dimensions sine qua non à une bonne compréhension du concept étudié. L'extension du didactiel à la matière abordée lors du cycle supérieur du primaire, par exemple, ne poserait aucun problème puisqu'il suffirait de rajouter de nouvelles occurences aux trois axes que nous avons défini.

D'autre part, l'analyse que nous avons réalisée préalablement à la conception de cet outil peut très bien s'appliquer aussi à tout autre domaine de l'E.A.O.. La méthode que nous avons suivie peut en effet s'adapter facilement à un logiciel traitant de géographie pour l'enseignement secondaire, les options que nous avons prises n'étant dues qu'à l'intérêt porté par l'instituteur avec lequel nous avons collaboré.

Le dernier point fort de ce didactiel que nous tenons à souligner se situe au niveau de la remédiation et du suivi des élèves. Grâce à une typologie des erreurs fréquemment commises - typologie que l'on peut naturellement étendre selon les besoins -, ainsi que de dossiers résumant les antécédents des différents étudiants, cette approche, rarement abordée, a pu être envisagée. Celle-ci a d'ailleurs suscité énormément d'intérêt parmi le personnel enseignant en raison du scepticisme accueillant cette démarche. Cependant, alors que le suivi des élèves a correspondu à l'attente qu'on s'en faisait, la remédiation ne fut pas, quant à elle, aussi probante que nous aurions pu l'espérer. Elle constitue néanmoins un domaine qui méritait de s'y attarder et qui mérite d'ailleurs toujours une étude approfondie.

Les extensions

La première extension envisageable se propose par rapport à la lacune existante, à savoir améliorer la remédiation.

Dans cette même optique, l'élaboration d'un véritable système d'intelligence artificielle pourrait être bénéfique à cette application. La démarche que nous avons suivie ne peut en effet être considérée comme la réalisation d'un système expert mais plutôt d'un système exploitant une base de connaissances composée de règles de production.

Cette nouvelle approche consisterait donc à améliorer l'interface-utilisateur, le système cognitif et la base de connaissances que nous avons développés durant ce travail. Ces extensions, par exemple permettraient à l'étudiant de diriger le dialogue; elles faciliteraient l'extension de la base de connaissances en réalisant une vérification syntaxique et sémantique des informations la composant; elles ajouteraient à cette même base de connaissances un ensemble de règles explicites résumant la stratégie à adopter alors qu'actuellment celle-ci est contenue implicitement dans le système cognitif et dans l'ordonnancement des règles; elles consisteraient à affiner le modèle que le système se fait de l'étudiant et permettraient ainsi d'améliorer le suivi des élèves; ...

Nous n'avons pas la prétention d'affirmer que l'Intelligence Artificielle nous offre toutes les solutions à la construction de bons didactiels. Cependant, nous estimons que cette nouvelle approche peut apporter beaucoup à ce domaine.

Enfin, nous serions heureux que ce travail puisse susciter la réalisation d'autres didactiels portant sur des exercices d'imitation, d'ajustement ou d'achèvement de l'initiative.

A P P E N D I C E 1

TEXTE DE LA PROPOSITION DE DECRET DE LA COMMISSION DE L'EDUCATION ET DE LA RECHERCHE SCIENTIFIQUE DE LA COMMUNAUTE FRANCAISE

ARTICLE 1

Dans l'enseignement primaire, les élèves seront sensibilisés à l'informatique par l'enseignement assisté par ordinateur.

L'accent sera mis sur la pratique élémentaire des outils informatiques appliquée aux problèmes concrets tirés des programmes généraux.

ARTICLE 2

§1. Dans l'enseignement secondaire, l'enseignement assisté par ordinateur permettra de compléter l'information reçue dans l'enseignement primaire. Aux 2e et 3e degrés, des notions d'informatique seront introduites dans les programmes de physique et de mathématiques. Les exercices seront assurés dans le cadre des programmes existants de toutes les disciplines où l'utilisation de l'ordinateur se justifie.

§2. Au 3e degré de l'enseignement secondaire, un cours d'informatique de deux heures par semaine est inscrit dans le cadre des options complémentaires.

A partir du 2e degré, dans le cadre de certaines options groupées, un cours d'informatique peut être organisé.

§3. Les dispositions visées au §2 du présent article sont également d'application dans les degrés ou sections d'études de l'enseignement secondaire traditionnel (type II).

ARTICLE 3

L'Exécutif organise des journées d'études afin d'assurer la formation ou le recyclage des enseignants appelés à enseigner les notions d'informatique ou à utiliser l'ordinateur dans le cadre de leurs cours.

ARTICLE 4

L'achat de matériel se fera en fonction du développement des programmes de formation ou de recyclage des enseignants et de fabrication de didactiels ainsi qu'en fonction de l'évolution de la technique de base.

ARTICLE 5

L'Exécutif de la Communauté française assure la mise en application du présent décret, en tenant compte de l'effectif disponible des enseignants formés ou recyclés, des didactiels réalisés dans ce but et des crédits affectés à cette activité.

ARTICLE 6

L'Exécutif de la Communauté française fixe la date d'entrée en vigueur du présent projet de décret.

Amendement proposé le 16 octobre 1984.

Au § 1er de l'article 2, supprimer :
"Aux 2e et 3e degrés, des notions d'informatique seront introduites dans les programmes de physique et de mathématiques. Les exercices seront assurés dans le cadre des programmes existants de toutes les disciplines où l'utilisation de l'ordinateur se justifie".

Justification.

L'amendement rejoint l'avis du Conseil d'Etat qui relève que cette disposition excède les limites de la compétence du Conseil de la Communauté française.

A P P E N D I C E 2

Dans ce tableau, nous évoquons le détail de la difficulté 3 mentionnée au chapitre 3. Comme nous l'avons remarqué, sa conception fut assez délicate en raison de son interdépendance avec les deux premières.
Cette difficulté concrétise, en fait, les quatre possibilités maximales de découpage d'une figure en parties équivalentes (de part leur forme). Comme tous les cas présentés n'admettent pas quatre possibilités, nous admettrons que, dans cette alternative, la dernière est répétée autant de fois que nécessaire.

1. CARRE

- Division en deux parties équivalentes.

    1. une médiane verticale.
    2. une médiane horizontale.
    3. une diagonale.

- Division en trois parties équivalentes.

    1. deux droites verticales.
    2. deux droites horizontales.

- Division en quatre parties équivalentes.

    1. deux médianes.
    2. trois droites verticales.
    3. trois droites horizontales.
    4. deux diagonales.

- Division en cinq parties équivalentes.

    1. quatre droites verticales.
    2. quatre droites horizontales.

- Division en six parties équivalentes.

    1. deux droites verticales et une droite horizontale.
    2. une droite verticale et deux droites horizontales.
    3. cinq droites verticales.
    4. cinq droites horizontales.

- Division en huit parties équivalentes.

    1. deux diagonales et deux médianes.
    2. trois droites verticales et une horizontale.
    3. une droite verticale et trois droites horizontales.

- Division en dix parties équivalentes.

    1. quatre verticales et une horizontale.
    2. une verticale et quatre horizontales.

- Division en douze parties équivalentes.

    1. trois verticales et deux horizontales.
    2. deux verticales et trois horizontales.

2. CARRE SUR POINTE

- Division en deux parties équivalentes.

    1. une diagonale.
    2. une médiane horizontale.
    3. une médiane verticale.

- Division en trois parties équivalentes.

    1. deux droites horizontales.
    2. deux droites verticales.

- Division en quatre parties équivalentes.

    1. deux diagonales.
    2. trois droites horizontales.
    3. trois droites verticales.
    4. deux médianes.

- Division en cinq parties équivalentes.

    1. quatre droites horizontales.
    2. quatre droites verticales.

- Division en six parties équivalentes.

    1. cinq droites horizontales.
    2. cinq droites verticales.
    3. une droite verticale et deux droites horizontales.
    4. deux droites verticales et une droite horizontale.

- Division en huit parties équivalentes.

    1. une droite verticale et trois droites horizontales
    2. trois droites verticales et une droite horizontale.
    3. deux diagonales et deux médianes.

- Division en dix parties équivalentes.

    1. une verticale et quatre horizontales.
    2. quatre verticales et une horizontale.

- Division en douze parties équivalentes.

    1. deux verticales et trois horizontales.
    2. trois verticales et deux horizontales.

3. RECTANGLE HORIZONTAL

- Division en deux parties équivalentes.

    1. une médiane verticale.
    2. une médiane horizontale.
    3. une diagonale.

- Division en trois parties équivalentes.

    1. deux droites verticales.
    2. deux droites horizontales.

- Division en quatre parties équivalentes.

    1. deux médianes.
    2. trois droites verticales.
    3. trois droites horizontales.

- Division en cinq parties équivalentes.

    1. quatre droites verticales.
    2. quatre droites horizontales.

- Division en six parties équivalentes.

    1. deux droites verticales et une droite horizontale.
    2. une droite verticale et deux droites horizontales.
    3. cinq droites verticales.
    4. cinq droites horizontales.

- Division en huit parties équivalentes.

    1. trois droites verticales et une horizontale.
    2. une droite verticale et trois droites horizontales.

- Division en dix parties équivalentes.

    1. quatre verticales et une horizontale.
    2. une verticale et quatre horizontales.

- Division en douze parties équivalentes.

    1. trois verticales et deux horizontales.
    2. deux verticales et trois horizontales.

4. RECTANGLE VERTICAL

- Division en deux parties équivalentes.

    1. une médiane verticale.
    2. une médiane horizontale.
    3. une diagonale.

- Division en trois parties équivalentes.

    1. deux droites verticales.
    2. deux droites horizontales.

- Division en quatre parties équivalentes.

    1. deux médianes.
    2. trois droites verticales.
    3. trois droites horizontales.

- Division en cinq parties équivalentes.

    1. quatre droites verticales.
    2. quatre droites horizontales.

- Division en six parties équivalentes.

    1. deux droites verticales et une droite horizontale.
    2. une droite verticale et deux droites horizontales.
    3. cinq droites verticales.
    4. cinq droites horizontales.

- Division en huit parties équivalentes.

    1. trois droites verticales et une horizontale.
    2. une droite verticale et trois droites horizontales.

- Division en dix parties équivalentes.

    1. quatre verticales et une horizontale.
    2. une verticale et quatre horizontales.

- Division en douze parties équivalentes.

    1. trois verticales et deux horizontales.
    2. deux verticales et trois horizontales.

5. CERCLE

Le cercle peut être découpé en parties équivalentes selon toutes les possibilités que nous proposons, à savoir en deux, trois, quatre, cinq, six, huit, dix et douze. Cette division est unique puisqu'elle utilise le principe des "quartiers".

6. TRIANGLE EQUILATERAL

- Division en deux parties équivalentes.

    1. une hauteur.
    2. une médiane.

7. HEXAGONE

- Division en deux parties équivalentes.

    1. une médiane verticale.
    2. une médiane horizontale.
    3. une diagonale.

- Division en trois parties équivalentes.

    1. trois demi-droites au sommet.

- Division en quatre parties équivalentes.

    1. une diagonale et une médiane.

- Division en six parties équivalentes.

    1. trois diagonales.
    2. trois médianes.

- Division en douze parties équivalentes.

    1. trois diagonales et trois médianes.

8. OCTOGONE

- Division en deux parties équivalentes.

    1. une diagonale.
    2. une médiane.

- Division en quatre parties équivalentes.

    1. deux diagonales.
    2. deux médianes.

- Division en huit parties équivalentes.

    1. quatre diagonales.
    2. quatre médianes.

A P P E N D I C E 3

Ce tableau reprend l'ensemble de la découpe en niveaux de difficultés que nous avons évoquée au chapitre 3. Celle-ci fut réalisée avec un instituteur : nous n'affirmons pas qu'elle est unique ni exhaustive. Porter une modification à cette découpe ne susciterait d'ailleurs pas d'importantes modifications au texte Pascal résultant de ces choix.

1. Difficulté relative au moyen de représentation utilisé.

- NIVEAU 1 : le carré.
- NIVEAU 2 : le carré sur pointe.
- NIVEAU 3 : le rectangle horizontal.
- NIVEAU 4 : le rectangle vertical.
- NIVEAU 5 : le cercle.
- NIVEAU 6 : le triangle équilatéral.
- NIVEAU 7 : l'hexagone.
- NIVEAU 8 : l'octogone.
- NIVEAU 9 : le segment.

2. Difficulté due au nombre de parties équivalentes dessinées sur le support graphique.

- NIVEAU 1 : 2 parties équivalentes.
- NIVEAU 2 : 4 parties équivalentes.
- NIVEAU 3 : 3 parties équivalentes.
- NIVEAU 4 : 6 parties équivalentes.
- NIVEAU 5 : 8 parties équivalentes.
- NIVEAU 6 : 5 parties équivalentes.
- NIVEAU 7 : 10 parties équivalentes.
- NIVEAU 8 : 12 parties équivalentes.

3. Difficulté née de la manière dont ces parties équivalentes sont représentées.

- NIVEAU 1 : première possibilité de découpage de la figure.
- NIVEAU 2 : deuxième possibilité de découpage de la figure.
- NIVEAU 3 : troisième possibilité de découpage de la figure.
- NIVEAU 4 : quatrième possibilité de découpage de la figure.

4. Difficulté résultant de la question posée à l'élève.

- NIVEAU 1 : trouver le nombre de parties équivalentes.
- NIVEAU 2 : trouver le nombre de parties équivalentes puis le convertir en fraction.
- NIVEAU 3 : associer une fraction à une des surfaces.
- NIVEAU 4 : associer "n" fraction à "n" des surfaces.
- NIVEAU 5 : trouver le nombre de parties équivalentes et de parties coloriées.
- NIVEAU 6 : trouver le nombre de parties équivalentes et de parties coloriées et ensuite, convertir ces nombres en une fraction.
- NIVEAU 7 : représenter une fraction dans une surface.
- NIVEAU 8 : identifier deux fractions et les relier par les signes "<", ">" ou "=".
- NIVEAU 9 : placer des fractions sur une droite graduée.
- NIVEAU 10 : trouver "x" fractions égales à "y".
- NIVEAU 11 : réaliser une opération et simplifier le résultat.

5. Difficulté due au nombre de figures soumises à l'élève.

- NIVEAU 1 : 1 figure identique.
- NIVEAU 2 : 2 figures identiques.
- NIVEAU 3 : 3 figures identiques.
- NIVEAU 4 : 4 figures identiques.
- NIVEAU 5 : 2 figures différentes.
- NIVEAU 6 : 3 figures différentes.
- NIVEAU 7 : 4 figures différentes.

6. Difficulté prenant en considération la nature des données.

- NIVEAU 1 : 1 fraction plus petite que l'unité dont le numérateur vaut 1.
- NIVEAU 2 : 1 fraction quelconque plus petite que l'unité.
- NIVEAU 3 : 2 fractions de même dénominateur.
- NIVEAU 4 : 2 fractions de même numérateur.
- NIVEAU 5 : 2 fractions dont l'une à son dénominateur multiple de l'autre.
- NIVEAU 6 : 1 fraction équivalente à 1 naturel.
- NIVEAU 7 : 2 fractions plus grandes que l'unité.
- NIVEAU 8 : 1 fraction équivalente à une autre.
- NIVEAU 9 : 2 fractions plus petites que l'unité de même dénominateur.
- NIVEAU 10 : 1 naturel et 1 fraction.
- NIVEAU 11 : 2 naturels et 1 fraction.
- NIVEAU 12 : 1 naturel et 2 fractions.
- NIVEAU 13 : 2 naturels et 2 fractions.
- NIVEAU 14 : 1 naturel et 1 fraction.
- NIVEAU 15 : 1 fraction et 1 naturel.

7. Difficulté de l'opération à effectuer.

- NIVEAU 1 : la simplification.
- NIVEAU 2 : l'addition.
- NIVEAU 3 : la soustraction.
- NIVEAU 4 : la multiplication.
- NIVEAU 5 : la division.

A P P E N D I C E 4

Ce tableau reprend la spécification des treize canevas proposés aux élèves. Pour chacun d'eux, nous proposons une "modulation" par les différentes difficultés.
Sa mise au point résulte d'une collaboration avec un instituteur; toutefois, toute modification est réalisable et n'engendrerait aucune perturbation.

CANEVAS 1 :

Il introduira le concept de "dénominateur d'une fraction". On proposera à l'élève une figure (disque, rectangle, polygône régulier, segment) divisé en un certain nombre de parties équivalentes et on lui posera diverses questions à ce sujet.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 1 à 9.
- Difficulté 2 : niveau 1 à 8.
- Difficulté 3 : niveau 1 à 4.
- Difficulté 4 : niveau 1 à 2.
- Difficulté 5 : niveau 1.
- Difficulté 6 : niveau 1.
- Difficulté 7 : niveau 0.

CANEVAS 2 :

Il introduira le concept de "dénominateur d'une fraction". On affichera plusieurs figures, et on fournira une ou plusieurs fractions que l'élève devra associer. (Ces figures seront identiques au début, quelconques ensuite, mais toutes seront divisées de manière différente).

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 1 à 9.
- Difficulté 2 : niveau 1 à 8.
- Difficulté 3 : niveau 1 à 4.
- Difficulté 4 : niveau 3 à 4.
- Difficulté 5 : niveau 2 à 7.
- Difficulté 6 : niveau 1.
- Difficulté 7 : niveau 0.

CANEVAS 3 :

Il introduira le concept de "numérateur d'une fraction". On affichera une figure divisée en un certain nombre de parties équivalentes, et on demandera à l'élève de trouver ce nombre. Ensuite, on coloriera plusieurs de celles-ci et on posera à l'élève diverses questions à ce sujet.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 1 à 9.
- Difficulté 2 : niveau 1 à 8.
- Difficulté 3 : niveau 1 à 4.
- Difficulté 4 : niveau 5 à 6.
- Difficulté 5 : niveau 1.
- Difficulté 6 : niveau 2.
- Difficulté 7 : niveau 0.

CANEVAS 4 :

Il introduira également le concept de "numérateur d'une fraction". On affichera plusieurs figures divisées en un même nombre de parties équivalentes, mais le nombre de ces parties équivalentes coloriées étant différent. On fournira ensuite une ou plusieurs fractions que l'élève devra associer.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 1 à 9.
- Difficulté 2 : niveau 2 à 8.
- Difficulté 3 : niveau 1 à 4.
- Difficulté 4 : niveau 3 à 4.
- Difficulté 5 : niveau 2 à 7.
- Difficulté 6 : niveau 3.
- Difficulté 7 : niveau 0.

CANEVAS 5 :

Il terminera l'introduction du concept de "numérateur et dénominateur d'une fraction". Il permettra à l'élève de représenter une fraction dans une surface.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 1 à 9.
- Difficulté 2 : niveau 2 à 8.
- Difficulté 3 : niveau 1 à 4.
- Difficulté 4 : niveau 7.
- Difficulté 5 : niveau 1.
- Difficulté 6 : niveau 1 à 2.
- Difficulté 7 : niveau 0.

CANEVAS 6 :

Il introduira le concept de "comparaison de fractions". On proposera deux surfaces identiques partagées en un certain nombre de parties équivalentes, certaines de ces parties étant hachurées. L'élève devra identifier les parties coloriées de ces figures et trouver si la première est plus petite, égale ou plus grande que la seconde.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 2 à 9.
- Difficulté 2 : niveau 1 à 8.
- Difficulté 3 : niveau 1 à 5.
- Difficulté 4 : niveau 8.
- Difficulté 5 : niveau 2.
- Difficulté 6 : niveau 3 à 8.
- Difficulté 7 : niveau 0.

CANEVAS 7 :

Il introduira également le concept de "comparaison de fractions". On proposera plusieurs fractions que l'élève devra placer sur une droite graduée.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 0.
- Difficulté 2 : niveau 2 à 8.
- Difficulté 3 : niveau 0.
- Difficulté 4 : niveau 9.
- Difficulté 5 : niveau 2 à 4.
- Difficulté 6 : niveau 3 à 7.
- Difficulté 7 : niveau 0.

CANEVAS 8 :

Il introduira le concept de "comparaison de fractions et celui de simplification de fractions". On proposera à l'élève diverses fractions que celui-ci devra simplifier au maximum.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 0.
- Difficulté 2 : niveau 2 à 8.
- Difficulté 3 : niveau 0.
- Difficulté 4 : niveau 11.
- Difficulté 5 : niveau 1.
- Difficulté 6 : niveau 6 à 8.
- Difficulté 7 : niveau 1.

CANEVAS 9 :

Il terminera l'introduction du concept de "comparaison de fractions". On proposera une fraction, et l'élève devra trouver une ou plusieurs fractions lui étant équivalentes.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 0.
- Difficulté 2 : niveau 2 à 8.
- Difficulté 3 : niveau 0.
- Difficulté 4 : niveau 10.
- Difficulté 5 : niveau 1 à 4.
- Difficulté 6 : niveau 6 à 8.
- Difficulté 7 : niveau 0.

CANEVAS 10 :

Il introduira le concept "d'addition et de soustraction de fractions". On proposera à l'élève plusieurs figures : la première représentant la fraction à additionner ou à soustraire; la dernière représentant l'opération effectuée. L'élève devra donc additionner ou soustraire les fractions et simplifier ensuite la réponse au maximum.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 1 à 9.
- Difficulté 2 : niveau 1 à 8.
- Difficulté 3 : niveau 1 à 5.
- Difficulté 4 : niveau 11.
- Difficulté 5 : niveau 2.
- Difficulté 6 : niveau 9 à 13.
- Difficulté 7 : niveau 2 à 3.

CANEVAS 11 :

Il terminera l'introduction du concept "d'addition et de soustraction de fractions". On proposera deux fractions que l'élève devra, soit additionner, soit soustraire, et dont il devra simplifier le résultat.

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 0.
- Difficulté 2 : niveau 1 à 8.
- Difficulté 3 : niveau 0.
- Difficulté 4 : niveau 11.
- Difficulté 5 : niveau 2.
- Difficulté 6 : niveau 9 à 13.
- Difficulté 7 : niveau 2 à 3.

CANEVAS 12 :

Il introduira le concept de "multiplication et de division de fractions". (cfr canevas 10).

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 1 à 9.
- Difficulté 2 : niveau 1 à 8.
- Difficulté 3 : niveau 1 à 5.
- Difficulté 4 : niveau 11.
- Difficulté 5 : niveau 2.
- Difficulté 6 : niveau 14 à 15.
- Difficulté 7 : niveau 4 à 5.

CANEVAS 13 :

Il terminera l'introduction du concept de "multiplication et de division de fractions". (cfr canevas 11).

Cet exercice sera modulé sur :

- Difficulté 1 : niveau 0.
- Difficulté 2 : niveau 1 à 8.
- Difficulté 3 : niveau 0.
- Difficulté 4 : niveau 11.
- Difficulté 5 : niveau 2.
- Difficulté 6 : niveau 14 à 15.
- Difficulté 7 : niveau 4 à 5.

B I B L I O G R A P H I E

[ANDR et alt.] : Andrianne et Dupagne.

Mathématiques en quatrième primaire.
Edition Labor.

[BERT, 1984] : Bertouille.

Formation informatique aujourd'hui et demain. Comment s'y préparer ?
Discours d'ouverture lors du colloque à la Hulpe du club Athéna.

[BODA et alt., 1983] : Bodart François et Pigneur Yves.

Modèles, outils et méthodes d'aide au développement d'un système d'information. Première partie : étude d'opportunité et analyse conceptuelle.

[BORK, 1984] : Bork Alfred.

Computers and the future : education.
dans : Comput, Educ. Vol. 8, num. 1, 1984. pp 1-4.

[CHOR, 1984] : Chorover Stephan L.

Cautions on computers in education.
dans : Byte. Juny 1984. pp 223-226.

[CLAN, 1983] : Clancey W.J.

Guidon.
dans : Journal of computer-based instruction. Vol. 10, summer 1983.

[CLAN et alt., 1984] : Clancey W.J. et Shortliffe E.M.

Intelligent computer-aided instruction for medical diagnosis.
dans : Readings in medical artificial intelligence.
Addison-Wesley Publishing Company. 1984. Chapter 11.

[COUT, 1982] : Coutty Marc.

Informatique matin, midi ... et soir !
Edition Autrement. Fevrier 1982.

[DANA, 1983] : Dana S.

Expert computer systems.
dans : IEEE, february 1983. pp 63-84.

[DUDA et alt., 1981] : Duda Richard et Gaschnig John.

Knowledge-based expert systems come of age.
dans : Byte, september 1981. pp 238-278.

[DUFO, 1985] : Dufour Pierre.

L'enseignement assisté par ordinateur.
dans : Les parents et l'école. pp 6-7.
Journal du CNAP (Confédération Nationale des Associations de Parents),
num. 2, mars-avril 1985.

[FAPEO, 1984] : Fédération nationale des Associations de Parents d'élèves de l'Enseignement Officiel.

L'informatique à l'école.
Editeur responsable : Quenon J. Bruxelles, novembre 1984.

[FERB, 1984] : Ferber J.

Les systèmes experts : un moteur d'inférence en pascal.
dans : Micro-Systèmes, mai 1984. pp 132-137.

[HAYE, 1984] : Hayes-Roth Frederick.

The knowledge-based expert system : a tutorial.
dans : IEEE, september 1984. pp 11-28.

[LARV, 1985] : Larvet Philippe.

Un moteur d'inférence d'ordre zéro.
dans : Micro-Systèmes, février 1985. pp 191-197.

[LAVR, 1984] : Laurent Jean-Pierre.

La structure de contrôle dans les systèmes experts.
dans : Technique et Science Informatique.
Vol. 3, num. 3, mars 1984. pp 161-177.

[MART et alt., 1982] : Martelli Alberto et Montanari Ugo.

An efficient unification algorithm.
dans : ACM Transactions on Programming Languages.
Vol. 4, num. 2, april 1982. pp 258-282.

[MIRA, 1981] : Mirabail Michel.

Les 50 mots clefs de la télématique.
Edition Privat. 1981.

[MONDE, 1982] : Journalistes du journal Le Monde.

L'informatique aujourd'hui. September 1982.

[PINS, 1981] : Pinson Suzanne.

Représentation des connaissances dans les systèmes experts.
dans : RAIRO Informatique Computer Science.
Vol. 15, num. 4, 1981. pp 343-367.

[POSW, 1984] : Poswick R.F.

Education en informatique.
dans : Journal de réflexion sur l'informatique.
Num. 1, mars 1985. pp 1-3

[PROGA] : Programat 4A.

Mathématiques en quatrième primaire.
Edition De Boeck.

[PROGB] : Programat 4B.

Mathématiques en quatrième primaire.
Edition De Boeck.

[ROBE, 1981] : Roberts Steven K.

Artificial intelligence.
dans : Byte, september 1981. pp 164-178.

[ROBI, 1965] : Robinson John Adam.

A machine oriented logic based on the resolution principle.
dans : Journal of the ACM. Vol. 12, num. 1, 1965. pp 23-41.

[ROBI, 1968] : Robinson John Adam.

The generalized resolution principle.
dans : Machine intelligence 3. D. Mitchie editor. pp 77-94.
Edimburgh University Press. 1968.

[ROBI, 1971] : Robinson John Adam.

Computational logic : the unification computation.
dans : Machine intelligence 6. B. Mitchie and H. Geltzen editors.
Edimburgh University Press. 1971.

[SMIE et alt.] : Smiets, Van cutsem.

Mathématiques modernes en quatrième primaire.
Edition Nomath.

[SIMO, 1982] : Simonnet Dominique.

L'intelligence artificielle est déjà née.
dans : L'Express, avril 1982. pp 95-101.

[TAIT, 1984] : Tait Kenneth.

The building of a computer based teaching system.
dans : Comput. Educ. Vol. 8, num. 1, 1984. pp 15-19.

[THUN] : Thunus N.

Mathématiques en quatrième primaire.

[VANL, 1984] : Van Lamsweerde A.

Cours de génie logiciel. Namur 1984.

[VAZQ et alt., 1984] : Vazquez-Abad Jesus et Larose Réal.

Computers, adaptive teaching and operational learning systems.
dans : Comput. Educ. Vol. 8, num. 1, 1984. pp 27-29.

[WINS, 1984] : Winston Patrick Henry.

Artificial intelligence.
Addison-Wesley Publishing Company. February 1984.

Manuels APPLE :

- Apple Pascal : Language Reference Manual.
- Apple Pascal : Operating System Reference Manual.
- Apple II : Reference Manual.

ANNEXE 1

```
procedure AFFICH(typesur:integer;coord1,coord2:integer ;
                 var param1,param2:integer;var absc,ordo:tableau) ;

    var dep1,dep2,i : integer ;
        cotes       : integer ;


    procedure PAFFICH(n:integer;l:integer) ;

        var i : integer ;

        begin
          pencolor(white) ;
          for i := 1 to n do
            begin
              absc[i] := turtlex ;
              ordo[i] := turtley ;
              move(l) ;
              turn (360 div n) ;
            end;
          pencolor(none);
        end;


    procedure RAFFICH(lg:integer;la:integer) ;

        var i : integer ;

        begin
          pencolor(white) ;
          i := 1 ;
          while (i < 5) do
            begin
              absc[i] := turtlex ;
              ordo[i] := turtley ;
              move(lg) ;
              turn(90) ;
              absc[i+1] := turtlex ;
              ordo[i+1] := turtley ;
              move(la) ;
              turn(90) ;
              i := i + 2 ;
            end ;
          pencolor(none) ;
        end ;


    procedure TRACERCLE(a,b,r:integer) ;

        var i : integer ;

        begin
          for i:=1 to 360 do
            begin
              turn(1) ;
              pencolor(none) ;
              moveto(a,b) ;
              move(r - 1) ;
              pencolor(white) ;
```

```
            move(1) ;
          end ;
      end ;


begin
  absc[0] := coord1 ;
  ordo[0] := coord2 ;
  dep1 := coord1 ;
  dep2 := coord2 ;
  case typesur of
    1 : begin
          param2 := param1 ;
          param1 := param1 + (param1 div 7) ;
          dep1 := coord1 + (param2 div 2) ;
          dep2 := coord2 - (param1 div 2) ;
          moveto (dep1,dep2) ;
          turn (90) ;
          raffich(param1,param2) ;
        end;
    2 : begin
          dep1 := coord1 + round((param1 * 1.414) / 2) ;
          moveto(dep1,dep2) ;
          absc[1] := turtlex ;
          ordo[1] := turtley ;
          turn (135) ;
          pencolor(white) ;
          move(param1) ;
          absc[2] := turtlex ;
          ordo[2] := turtley ;
          turn(95) ;
          move(param1) ;
          absc[3] := turtlex ;
          ordo[3] := turtley ;
          turn(85) ;
          move(param1) ;
          absc[4] := turtlex ;
          ordo[4] := turtley ;
          turn(95) ;
          move(param1) ;
          pencolor(none) ;
        end;
    3 : begin
          dep1 := coord1 + (param1 div 2) ;
          dep2 := coord2 - (param2 div 2) ;
          moveto(dep1,dep2) ;
          turn(90) ;
          raffich(param2,param1) ;
        end ;
    4 : begin
          dep1 := coord1 + (param2 div 2) ;
          dep2 := coord2 - (param1 div 2) ;
          moveto(dep1,dep2) ;
          turn(90) ;
          raffich(param1,param2) ;
        end ;
    5 : tracercle(coord1,coord2,param1) ;
    6 : begin
          dep2 := coord2 + round(param1 / 1.732) ;
          moveto(dep1,dep2) ;
          turn(240) ;
```

```
            cotes := 3 ;
            paffich(cotes,param1) ;
          end ;
      7 : begin
            dep1 := coord1 + param1 ;
            moveto(dep1,dep2) ;
            turn(120) ;
            cotes := 6 ;
            paffich(cotes,param1) ;
          end ;
      8 : begin
            dep1 := coord1 + round(param1 / 0.765) ;
            moveto(dep1,dep2) ;
            turn(112) ;
            cotes := 8 ;
            paffich(cotes,param1) ;
          end ;
      9 : begin
            dep1 := coord1 - (param1 div 2) ;
            moveto(dep1,dep2) ;
            pencolor(white) ;
            absc[1] := turtlex ;
            ordo[1] := turtley ;
            move(param1) ;
            absc[2] := turtlex ;
            ordo[2] := turtley ;
            pencolor(none) ;
          end ;
    end ;
  end ;
```

```
procedure PARTAGE(typesur:integer;param1,param2:integer;nbrepar:integer;
                  diffpar:integer;absc,ordo:tableau ) ;


  function NBRESTS : integer ;

    begin
      case typesur of
        1,2,3,4 : nbrests:=4 ;
        6       : nbrests:=3 ;
        7       : nbrests:=6 ;
        8       : nbrests:=8 ;
      end ;
    end ;


  function MODU(nbrtest:integer;borne:integer) : integer ;

    begin
      if (nbrtest > borne) then modu:=nbrtest - borne
                           else modu:=nbrtest ;
    end ;
```

```
procedure TVPART(long:integer;n:integer) ;

    var pas,i : integer ;

    begin
      pas := round(long/n) ;
      for i:=0 to n do
        begin
          moveto((absc[1]+(i*pas)),ordo[1]) ;
          turn(90) ;
          move(3) ;
          turn(180) ;
          pencolor(white) ;
          move(7) ;
          pencolor(none) ;
          turn(90) ;
        end ;
    end ;


procedure DVPART(n:integer) ;

    var nbre,i : integer ;
        ecart1,ecart2,d1,d2,a1,a2 : integer ;
        numarr : integer ;

    begin
      nbre := nbrests ;
      ecart1 := round((absc[2]-absc[3])/n) ;
      ecart2 := round((ordo[2]-ordo[3])/n) ;
      for i:=1 to (n-1) do
        begin
          d1 := absc[2] - (ecart1 * i) ;
          d2 := ordo[2] - (ecart2 * i) ;
          moveto(d1,d2) ;
          numarr := modu((nbre div 2) + 3,nbre) ;
          a1 := absc[numarr] - (ecart1 * i) ;
          a2 := ordo[numarr] - (ecart2 * i) ;
          pencolor(white) ;
          moveto(a1,a2) ;
          pencolor(none) ;
        end ;
    end ;


procedure DHPART(n:integer) ;

    var nbre,i,numarr : integer ;
        ecart1,ecart2,d1,d2,a1,a2 : integer ;

    begin
      nbre := nbrests ;
      ecart1 := round((abs(absc[1]-absc[2]))/n) ;
      ecart2 := round((abs(ordo[1]-ordo[2]))/n) ;
      for i:= 1 to (n-1) do
        begin
          d1 := absc[1] - (ecart1 * i) ;
          d2 := ordo[1] + (ecart2 * i) ;
          moveto(d1,d2) ;
          numarr := (nbre div 2) + 2 ;
```

```
            a1 := absc[numarr] - (ecart1 * i) ;
            a2 := ordo[numarr] + (ecart2 * i) ;
            pencolor(white) ;
            moveto(a1,a2) ;
            pencolor(none) ;
          end ;
      end ;


procedure DPART(numsom:integer) ;

    var nbre,numarr : integer ;

    begin
      nbre := nbrests ;
      moveto(absc[numsom],ordo[numsom]) ;
      numarr := (nbre div 2) + numsom ;
      pencolor(white) ;
      moveto(absc[numarr],ordo[numarr]) ;
      pencolor(none) ;
    end ;


procedure MPART(numsom:integer) ;

    var nbre,numarr : integer ;
        mil1,mil2 : integer ;

    begin
      nbre := nbrests ;
      mil1 := round((absc[numsom]+(absc[modu((numsom+1),nbre)]))/2) ;
      mil2 := round((ordo[numsom]+(ordo[modu((numsom+1),nbre)]))/2) ;
      moveto(mil1,mil2) ;
      numarr := modu((nbre div 2) + numsom,nbre) ;
      mil1 := round((absc[numarr]+(absc[modu((numarr+1),nbre)]))/2) ;
      mil2 := round((ordo[numarr]+(ordo[modu((numarr+1),nbre)]))/2) ;
      pencolor(white) ;
      moveto(mil1,mil2) ;
      pencolor(none) ;
    end ;


procedure VPHPART(n:integer;var ndv,ndh:integer) ;

    begin
      case n of
        6 : begin
              ndv := 3 ;
              ndh := 2 ;
            end ;
        8 : begin
              ndv := 4 ;
              ndh := 2 ;
            end ;
        10: begin
              ndv := 5 ;
              ndh := 2 ;
            end ;
        12: if (random > 16384) then
```

```
                begin
                  ndv := 6 ;
                  ndh := 2
                end
                                    else
                begin
                  ndv := 4 ;
                  ndh := 3 ;
                end ;
            end ;
        dvpart(ndv) ;
        dhpart(ndh) ;
      end ;


procedure VMHPART(n:integer;var ndv,ndh:integer) ;

    begin
      case n of
        4 : begin
              ndv := 2 ;
              ndh := 2 ;
            end ;
        6 : begin
              ndv := 2 ;
              ndh := 3 ;
            end ;
        8 : begin
              ndv := 2 ;
              ndh := 4 ;
            end ;
        10: begin
              ndv := 2 ;
              ndh := 5 ;
            end ;
        12: if (random > 16384) then
              begin
                ndv := 2 ;
                ndh := 6
              end
                                    else
              begin
                ndv := 3 ;
                ndh := 4 ;
              end ;
            end ;
        dvpart(ndv) ;
        dhpart(ndh) ;
      end ;


procedure DMPART(n:integer) ;

    var i,nbredr : integer ;

    begin
      if (typesur = 1) or (typesur = 2) then
          begin
            vmhpart(n div 2,ndv,ndh) ;
            dpart(1) ;
```

```
          dpart(2)
        end
                                          else
        begin
          nbredr := n div 4 ;
          for i:=1 to nbredr do
            begin
              dpart(i) ;
              mpart(i+1) ;
            end ;
        end ;
    end ;


procedure HTPART ;

    var mil1,mil2 : integer ;

    begin
      moveto(absc[1],ordo[1]) ;
      mil1 := round((absc[3]-absc[2]) / 2) + absc[2] ;
      mil2 := ordo[2] ;
      pencolor(white) ;
      moveto(mil1,mil2) ;
      pencolor(none) ;
    end ;


procedure MTPART ;

    var mil1,mil2 : integer ;

    begin
      moveto(absc[2],ordo[2]) ;
      mil1 := round((absc[3] + absc[1]) / 2) ;
      mil2 := round((ordo[3] + ordo[1]) / 2) ;
      pencolor(white) ;
      moveto(mil1,mil2) ;
      pencolor(none) ;
    end ;


procedure DDPART(n:integer;rayon:integer) ;

    var i,numsom : integer ;

    begin
      if (typesur = 7) then
        begin
          numsom := 1 ;
          for i:=1 to 3 do
            begin
              moveto(absc[numsom],ordo[numsom]) ;
              pencolor(white) ;
              moveto(absc[0],ordo[0]) ;
              pencolor(none) ;
              numsom := numsom + 2
            end
        end
```

```
                    else
        begin
          pencolor(none) ;
          for i:= 1 to n do
            begin
              moveto(absc[0],ordo[0]) ;
              pencolor(white) ;
              move(rayon) ;
              pencolor(none) ;
              turn(360 div n) ;
            end ;
        end ;
    end ;


begin
  case typesur of
    1 : case nbrepar of
          2     : begin
                    if (diffpar = 1) then dvpart(nbrepar) ;
                    if (diffpar = 2) then dhpart(nbrepar) ;
                    if (diffpar > 2) then dpart(1) ;
                  end ;
          3,5   : if (diffpar = 1) then dvpart(nbrepar)
                                   else dhpart(nbrepar) ;
          4     : begin
                    if (diffpar = 1) then vmhpart(nbrepar,ndv,ndh) ;
                    if (diffpar = 2) then dvpart(nbrepar) ;
                    if (diffpar = 3) then dhpart(nbrepar) ;
                    if (diffpar > 3) then begin dpart(1);
                                                dpart(2);
                                          end ;
                  end ;
          6     : begin
                    if (diffpar = 1) then vphpart(nbrepar,ndv,ndh) ;
                    if (diffpar = 2) then vmhpart(nbrepar,ndv,ndh) ;
                    if (diffpar = 3) then dvpart(nbrepar) ;
                    if (diffpar > 3) then dhpart(nbrepar) ;
                  end ;
          8     : begin
                    if (diffpar = 1) then dmpart(nbrepar) ;
                    if (diffpar = 2) then vphpart(nbrepar,ndv,ndh) ;
                    if (diffpar > 2) then vmhpart(nbrepar,ndv,ndh) ;
                  end ;
          10,12 : if (diffpar = 1) then vphpart(nbrepar,ndv,ndh)
                                   else vmhpart(nbrepar,ndv,ndh) ;
        end ;
    3,4:case nbrepar of
          2     : begin
                    if (diffpar = 1) then dvpart(nbrepar) ;
                    if (diffpar = 2) then dhpart(nbrepar) ;
                    if (diffpar > 2) then dpart(1) ;
                  end ;
          3,5   : if (diffpar = 1) then dvpart(nbrepar)
                                   else dhpart(nbrepar) ;
          4     : begin
                    if (diffpar = 1) then vmhpart(nbrepar,ndv,ndh) ;
                    if (diffpar = 2) then dvpart(nbrepar) ;
                    if (diffpar > 2) then dhpart(nbrepar) ;
                  end ;
          6     : begin
```

```
                    if (diffpar = 1) then vphpart(nbrepar,ndv,ndh) ;
                    if (diffpar = 2) then vmhpart(nbrepar,ndv,ndh) ;
                    if (diffpar = 3) then dvpart(nbrepar) ;
                    if (diffpar > 3) then dhpart(nbrepar) ;
                  end ;
          8,10,12:if (diffpar = 1) then vphpart(nbrepar,ndv,ndh)
                                   else vmhpart(nbrepar,ndv,ndh) ;
        end ;
      5 : ddpart(nbrepar,param1) ;
      6 : if (diffpar = 1) then htpart
                           else mtpart ;
      7 : case nbrepar of
            2     : begin
                    if (diffpar = 1) then dvpart(nbrepar) ;
                    if (diffpar = 2) then dhpart(nbrepar) ;
                    if (diffpar > 2) then dpart(1) ;
                  end ;
            3     : ddpart(nbrepar,0) ;
            4,12  : dmpart(nbrepar) ;
            6     : if (diffpar = 1) then begin dpart(1) ;
                                                dpart(2) ;
                                                dpart(3)
                                          end
                                     else begin mpart(1) ;
                                                mpart(2) ;
                                                mpart(3) ;
                                          end ;
        end ;
      8 : case nbrepar of
            2     : if (diffpar = 1) then dpart(1)
                                     else mpart(1) ;
            4     : if (diffpar = 1) then
                    begin
                      dpart(1) ;
                      dpart(3)
                    end
                                     else
                    begin
                      mpart(1) ;
                      mpart(3) ;
                    end ;
            8     : if (diffpar = 1) then for i:=1 to 4 do dpart(i)
                                     else for i:=1 to 4 do mpart(i) ;
        end ;
      9 : tvpart(param1,nbrepar) ;
    end ;
end ;
```

```
procedure COLORIAGE(typesur:integer;param1,param2:integer;nbrepar:integer;
                    diffpar:integer;absc,ordo:tableau;nbcolor:integer) ;


   procedure COL4H(coord1,coord2:integer;long,larg:integer) ;

      var i : integer ;

      begin
        for i:=0 to (long - 1) do
            begin
              moveto(coord1,coord2 + i) ;
              pencolor(white) ;
              moveto(coord1 - larg,coord2 + i) ;
              pencolor(none) ;
            end ;
      end ;


   procedure DVCOLOR(n:integer) ;

      var long,i,j,step : integer ;
          coord1,coord2,larg : integer ;

      begin
        step := round((absc[2] - absc[3]) / nbrepar) ;
        if (typesur = 3) then long := param2 - 3
                         else long := param1 - 3 ;
        for j:=0 to (n-1) do
            begin
              coord1 := (absc[2] - (j * step)) - 2 ;
              coord2 := ordo[2] - 2 ;
              larg := step - 4 ;
              for i:=0 to (long - 1) do
                begin
                  moveto(coord1,coord2 - i) ;
                  pencolor(white) ;
                  moveto(coord1 - larg,coord2-i) ;
                  pencolor(none) ;
                end ;
            end ;
      end ;


   procedure DHCOLOR(n:integer) ;

      var larg,j,step : integer ;
          coord1,coord2,long : integer ;

      begin
        step := round((ordo[2] - ordo[1]) / nbrepar) ;
        if (typesur = 3) then larg := param1 - 4
                         else larg := param2 - 4 ;
        for j:=0 to (n-1) do
          begin
            coord1 := absc[1] - 2 ;
            coord2 := (ordo[1] + (j * step)) + 2 ;
            long := step - 3 ;
            col4h(coord1,coord2,long,larg) ;
```

```
        end ;
    end ;


procedure VHCOLOR(n:integer) ;

    var long,larg,step1,step2,k,j : integer ;
        fini : boolean ;
        coord1,coord2 : integer ;

    begin
      step1 := round((absc[2] - absc[3]) / ndv) ;
      step2 := round((ordo[2] - ordo[1]) / ndh) ;
      long := step2 - 3 ;
      larg := step1 - 4 ;
      k := 0 ;
      fini := false ;
      while (k < ndv) and (not fini) do
        begin
          j := 0 ;
          while (j < ndh) and (not fini) do
            begin
              coord1 := (absc[1] - 2) - (k * step1) ;
              coord2 := (ordo[1] + (j * step2)) + 2 ;
              col4h(coord1,coord2,long,larg) ;
              j := j + 1 ;
              if (n = ((k * ndh) + j)) then fini:=true ;
            end ;
          k := k + 1 ;
        end ;
    end ;


procedure CCOLOR(n:integer) ;

    var i,j : integer ;

    begin
      for j:=1 to n do
        begin
          moveto(absc[0],ordo[0]) ;
          turn(3) ;
          for i:=1 to ((360 div nbrepar) - 6) do
            begin
              moveto(absc[0],ordo[0]) ;
              pencolor(white) ;
              move(param1 - 3) ;
              pencolor(none) ;
              turn(1) ;
            end ;
          turn(3) ;
        end ;
    end ;
```

```
procedure CAR2COLOR(n:integer) ;

    var i : integer ;

    begin
      for i:=5 to (param2 - 2) do
        begin
          moveto(absc[1] - 2,ordo[1] + i) ;
          pencolor(white) ;
          moveto(absc[3] + i,ordo[3] - 2) ;
          pencolor(none) ;
        end ;
      if (n = 2) then
        for i:=5 to (param2 - 2) do
          begin
            moveto(absc[1] - i,ordo[1] + 2) ;
            pencolor(white) ;
            moveto(absc[3] + 2,ordo[3] - i) ;
            pencolor(none) ;
          end ;
    end ;


procedure CAR8COLOR (n:integer) ;


    var i : integer ;

    begin
      for i:=5 to ((param2 - 2) div 2) - 1 do
        begin
          moveto(absc[1] - 2,ordo[1] + i) ;
          pencolor(white) ;
          moveto(absc[0] + i ,ordo[0] - 2) ;
          pencolor(none) ;
        end ;
      if (n >= 2) then
        for i:=5 to ((param2 - 2) div 2) -1 do
          begin
            moveto(absc[0] + i,ordo[0] + 2) ;
            pencolor(white) ;
            moveto(absc[2] - 2,ordo[2] - i) ;
            pencolor(none) ;
          end ;
      if (n >= 3) then
        for i:=5 to ((param2 - 2) div 2) - 1 do
          begin
            moveto(absc[0] + 2,ordo[0] + i) ;
            pencolor(white) ;
            moveto(absc[2] - i,ordo[2] - 2) ;
            pencolor(none) ;
          end ;
      if (n >= 4) then
        for i:=5 to ((param2 - 2) div 2) - 1 do
          begin
            moveto(absc[0] - 2,ordo[0] + i) ;
            pencolor(white) ;
            moveto(absc[3] + i,ordo[3] - 2) ;
            pencolor(none) ;
          end ;
      if (n >= 5) then
```

```
        for i:=5 to ((param2 - 2) div 2) - 1 do
          begin
            moveto(absc[0] - i,ordo[0] + 2) ;
            pencolor(white) ;
            moveto(absc[3] + 2,ordo[3] - i) ;
            pencolor(none) ;
          end ;
        if (n >= 6) then
          for i:=5 to ((param2 - 2) div 2) - 1 do
          begin
            moveto(absc[4] + 2,ordo[4] + i) ;
            pencolor(white) ;
            moveto(absc[0] - i,ordo[0] - 2) ;
            pencolor(none) ;
          end ;
        if (n >= 7) then
          for i:=5 to ((param2 - 2) div 2) - 1 do
          begin
            moveto(absc[4] + i,ordo[4] + 2) ;
            pencolor(white) ;
            moveto(absc[0] - 2,ordo[0] - i) ;
            pencolor(none) ;
          end ;
        if (n >= 8) then
          for i:=5 to ((param2 - 2) div 2) - 1 do
          begin
            moveto(absc[1] - i,ordo[1] + 2) ;
            pencolor(white) ;
            moveto(absc[0] + 2,ordo[0] - i) ;
            pencolor(none) ;
          end ;
      end ;


procedure CAR4COLOR (n : integer) ;

    var i : integer ;

    begin
      for i:=5 to (param2 div 2) - 2 do
        begin
          moveto(absc[1] - 2,ordo[1] + i) ;
          pencolor(white) ;
          moveto(absc[0] + i ,ordo[0]) ;
          pencolor(none) ;
        end ;
      for i:=5 to (param2 div 2) - 2 do
        begin
          moveto(absc[0] + i,ordo[0]) ;
          pencolor(white) ;
          moveto(absc[2] - 2,ordo[2] - i) ;
          pencolor(none) ;
        end ;
      if (n >= 2) then
       begin
        for i:=5 to (param2 div 2) do
          begin
            moveto(absc[0],ordo[0] + i) ;
            pencolor(white) ;
            moveto(absc[2] - i,ordo[2] - 2) ;
            pencolor(none) ;
```

```
            end ;
          for i:=5 to (param2 div 2) do
            begin
              moveto(absc[0],ordo[0] + i) ;
              pencolor(white) ;
              moveto(absc[3] + i,ordo[3] - 2) ;
              pencolor(none) ;
            end ;
        end ;
       if (n >= 3) then
        begin
         for i:=5 to (param2 div 2) - 2 do
           begin
             moveto(absc[0] - i,ordo[0]) ;
             pencolor(white) ;
             moveto(absc[3] + 2,ordo[3] - i) ;
             pencolor(none) ;
           end ;
         for i:=5 to (param2 div 2) - 2 do
           begin
             moveto(absc[4] + 2,ordo[4] + i) ;
             pencolor(white) ;
             moveto(absc[0] - i,ordo[0]) ;
             pencolor(none) ;
           end ;
        end ;
       if (n >= 4) then
        begin
         for i:=5 to (param2 div 2) do
           begin
             moveto(absc[4] + i,ordo[4] + 2) ;
             pencolor(white) ;
             moveto(absc[0],ordo[0] - i) ;
             pencolor(none) ;
           end ;
         for i:=5 to (param2 div 2) do
           begin
             moveto(absc[1] - i,ordo[1] + 2) ;
             pencolor(white) ;
             moveto(absc[0],ordo[0] - i) ;
             pencolor(none) ;
           end ;
        end ;
    end ;


procedure TVCOLOR (n:integer) ;

    var i,j,pas : integer ;

    begin
      pas := round(param1 / nbrepar) ;
      for i:=0 to n-1 do
        for j:=1 to 7 do
          begin
            moveto((absc[1] + (i * pas) + 2),ordo[1] + j - 4) ;
            pencolor(white) ;
            move(pas - 4) ;
            pencolor(none) ;
          end ;
    end ;
```

```
procedure REC2COLOR(n:integer) ;

    var i : integer ;

    begin
      if (typesur = 3)
        then
          begin
            turn (90) ;
            for i:=2 to (param2 - 2) do
              begin
                moveto(absc[2] - 2,ordo[2] - i) ;
                pencolor(white) ;
                move(param1 - 5 - round((param1 / param2) * i)) ;
                pencolor(none) ;
              end ;
            if (n=2) then
              begin
                turn(180) ;
                for i:=2 to (param2 - 2) do
                  begin
                    moveto(absc[4] + 2,ordo[4] + i) ;
                    pencolor(white) ;
                    move(param1 - 5 - round((param1 / param2) * i)) ;
                    pencolor(none)
                  end
              end
          end
        else
          begin
            turn(180) ;
            for i:=2 to (param2 - 2) do
              begin
                moveto(absc[2] - i,ordo[2] - 2) ;
                pencolor(white) ;
                move(param1 - 5 - round((param1 / param2) * i)) ;
                pencolor(none) ;
              end ;
            if (n=2) then
              begin
                turn(180) ;
                for i:=2 to (param2 - 2) do
                  begin
                    moveto(absc[4] + i,ordo[4] + 2) ;
                    pencolor(white) ;
                    move(param1 - 5 - round((param1 / param2) * i)) ;
                    pencolor(none) ;
                  end ;
              end ;
          end ;
    end ;


procedure TRICOLOR (n:integer) ;

    var i,mil1,mil2 : integer ;

    begin
      if (diffpar = 1)
        then
```

```
            begin
              turn(210) ;
              for i:=1 to (param1 div 2) - 4 do
                begin
                  moveto(absc[2]+(param1 div 2)+1+i,ordo[2] + 2) ;
                  pencolor(white) ;
                  move((ordo[0] - ordo[2]) + round(param1 / 1.732) -
                        5 - round(1.732 * i)) ;
                end ;
              if (n=2) then
                for i:=1 to (param1 div 2) - 4 do
                  begin
                    moveto(absc[2]+(param1 div 2)-1-i,ordo[2] + 2) ;
                    pencolor(white) ;
                    move((ordo[0] - ordo[2]) + round(param1 / 1.732) -
                          5- round(1.732 * i)) ;
                    pencolor(none) ;
                  end
            end
          else
            begin
              mil1 := round((absc[3] + absc[1]) / 2) ;
              mil2 := round((ordo[3] + ordo[1]) / 2) ;
              turn(240) ;
              for i:=2 to (param1 div 2) - 4 do
                begin
                  moveto(mil1,mil2) ;
                  move(i) ;
                  turn(90) ;
                  pencolor(white) ;
                  move((ordo[0] - ordo[2]) + round(param1 / 1.732) -
                        4 - round(1.732 * i)) ;
                  pencolor(none) ;
                  turn(270) ;
                end ;
              if (n = 2) then
                begin
                  mil1 := round((absc[3] + absc[1]) / 2) ;
                  mil2 := round((ordo[3] + ordo[1]) / 2) ;
                  for i:=2 to (param1 div 2) - 2 do
                    begin
                      moveto(mil1,mil2) ;
                      move(-i) ;
                      turn(90) ;
                      pencolor(white) ;
                      move((ordo[0] - ordo[2]) + round(param1 / 1.732) -
                            4 - round(1.732 * i)) ;
                      pencolor(none) ;
                      turn(-90) ;
                    end ;
                end ;
            end ;
      end ;



begin
  case typesur of
    1 : case nbrepar of
          2   : begin
                  if (diffpar = 1) then dvcolor(nbcolor) ;
```

```
                    if (diffpar = 2) then dhcolor(nbcolor) ;
                    if (diffpar > 2) then car2color(nbcolor) ;
                  end ;
            3,5 : if (diffpar = 1) then dvcolor(nbcolor)
                                   else dhcolor(nbcolor) ;
            4   : begin
                    if (diffpar = 1) then vhcolor(nbcolor) ;
                    if (diffpar = 2) then dvcolor(nbcolor) ;
                    if (diffpar = 3) then dhcolor(nbcolor) ;
                    if (diffpar > 3) then car4color(nbcolor) ;
                  end ;
            6   : begin
                    if (diffpar = 1) then vhcolor(nbcolor) ;
                    if (diffpar = 2) then vhcolor(nbcolor) ;
                    if (diffpar = 3) then dvcolor(nbcolor) ;
                    if (diffpar > 3) then dhcolor(nbcolor) ;
                  end ;
            8   : begin
                    if (diffpar = 1) then car8color(nbcolor) ;
                    if (diffpar > 1) then vhcolor(nbcolor) ;
                  end ;
          10,12 : vhcolor(nbcolor) ;
           end ;
      3,4:case nbrepar of
            2     : begin
                      if (diffpar = 1) then dvcolor(nbcolor) ;
                      if (diffpar = 2) then dhcolor(nbcolor) ;
                      if (diffpar > 2) then rec2color(nbcolor) ;
                    end ;
            3,5   : if (diffpar = 1) then dvcolor(nbcolor)
                                     else dhcolor(nbcolor) ;
            4     : begin
                      if (diffpar = 1) then vhcolor(nbcolor) ;
                      if (diffpar = 2) then dvcolor(nbcolor) ;
                      if (diffpar > 2) then dhcolor(nbcolor) ;
                    end ;
            6     : begin
                      if (diffpar = 1) then vhcolor(nbcolor) ;
                      if (diffpar = 2) then vhcolor(nbcolor) ;
                      if (diffpar = 3) then dvcolor(nbcolor) ;
                      if (diffpar > 3) then dhcolor(nbcolor) ;
                    end ;
            8,10,12:vhcolor(nbcolor) ;
           end ;
      5  : ccolor(nbcolor) ;
      6  : tricolor(nbcolor) ;
      7  : hex12color(nbcolor) ;
      9  : tvcolor(nbcolor) ;
   end ;
end ;
```

ANNEXE 2

```
function RAND(low,high:integer):integer ;

    var c,mx,d : integer ;

    begin
      if (low = high) then rand := low
                      else
        begin
          c := high - low + 1 ;
          mx := (maxint - high + low) div c + 1 ;
          mx := mx * (high - low) + (mx - 1) ;
          repeat d:=random until d<=mx ;
          rand := low + d mod c ;
        end ;
    end ;



procedure FRACTGEN(diff7,diff8,niv2:integer;diff2:tabl;
                   var nat1,nat2,denom1,denom2,num1,num2:integer) ;

    var k : integer ;

    procedure MULTIPLE(var mul1,mul2:integer) ;

        var i,j2,j3,j5 : integer ;
            ok         : boolean ;
            m2         : array[1..6] of integer ;
            m3         : array[1..3] of integer ;
            m5         : array[1..2] of integer ;

        begin
          j2 := 0 ;
          j3 := 0 ;
          j5 := 0 ;
          for i:=1 to niv2 do
            begin
              if (diff2[i] mod 2 = 0) then begin j2 := j2 + 1 ;
                                                 m2[j2] := diff2[i] ;
                                           end ;
              if (diff2[i] mod 3 = 0) then begin j3 := j3 + 1 ;
                                                 m3[j3] := diff2[i] ;
                                           end ;
              if (diff2[i] mod 5 = 0) then begin j5 := j5 + 1 ;
                                                 m5[j5] := diff2[i] ;
                                           end ;
            end ;
          ok := false ;
          while (not ok) do
            begin
              mul1 := diff2[rand(1,niv2)] ;
              if ((mul1 mod 2 = 0) and (j2 > 1))
                then
                  begin
                    repeat mul2:=m2[rand(1,j2)] until (mul1 <> mul2);
                    ok := true ;
                  end ;
              if ((mul1 mod 3 = 0) and (j3 > 1))
                then
                  begin
```

```
                repeat mul2:=m3[rand(1,j3)] until (mul1 <> mul2);
                ok := true ;
              end ;
            if ((mul1 mod 5 = 0) and (j5 > 1))
              then
                begin
                repeat mul2:=m5[rand(1,j5)] until (mul1 <> mul2);
                ok := true ;
                end ;
          end ;
    end ;


begin
  nat1 := 0 ;
  nat2 := 0 ;
  denom2 := 0 ;
  num2 := 0 ;
  denom1 := diff2[rand(1,niv2)] ;
  num1 := rand(1,denom1 - 1) ;
  case diff7 of
    1         : num1 := 1 ;
    3,9       : begin
                  num2 := rand(1,denom1 - 1) ;
                  denom2 := denom1 ;
                end ;
    4         : begin
                  num2 := num1 ;
                  denom2 := diff2[rand(1,niv2)] ;
                end ;
    5         : begin
                  multiple(denom1,denom2) ;
                  num1 := rand(1,denom1 - 1) ;
                  num2 := rand(1,denom2 - 1) ;
                end ;
    6         : num1 := num1 * denom1 ;
    7         : num1 := rand(denom1 + 1,20) ;
    8         : begin
                  multiple(denom1,denom2) ;
                  if (denom1 < denom2) then
                    begin
                      num1 := rand(1,denom1 - 1) ;
                      num2 := num1 * (denom2 div denom1)
                    end
                                        else
                    begin
                      num2 := rand(1,denom2 - 1) ;
                      num1 := num2 * (denom1 div denom2) ;
                    end ;
                end ;
    10        : nat1 := rand(1,10) ;
    11        : if (diff8 = 3) then begin nat1 := rand(2,10) ;
                                           nat2 := rand(1,nat1 - 1)
                                    end
                                else
                                    begin nat1 := rand(1,10) ;
                                          nat2 := rand(1,10) ;
                                    end ;
    12        : begin
                  nat1 := rand(1,10) ;
                  denom2 := denom1 ;
                  num2 := rand(1,denom2 - 1) ;
```

```
            end ;
    13    : begin
              denom2 := denom1 ;
              num2 := rand(1,denom2 - 1) ;
              if (diff8 = 3)
                then
                  if (num1 < num2)
                    then
                      begin
                        nat1 := rand(2,10) ;
                        nat2 := rand(1,nat1 - 1)
                      end
                    else
                      begin
                        nat1 := rand(1,10) ;
                        nat2 := rand(1,nat1)
                      end
                else
                  begin
                    nat1 := rand(1,10) ;
                    nat2 := rand(1,10) ;
                  end ;
            end ;
    14    : begin
              if (diff8 <> 5)
                then nat1 := rand(1,10)
                else multiple(denom1,nat1) ;
              if (denom1 < nat1)
                then
                  begin
                    k := nat1 ;
                    nat1 := denom1 ;
                    denom1 := k ;
                  end ;
            end ;
    end ;
end ;
```

ANNEXE 3

```
procedure INREGLE(var tabin : tab1) ;

    var car : char ;
        k,nbrdig : integer ;

    begin
      for k:=1 to max1 do tabin[k]:=0 ;
      write('     ') ;
      k := 0 ;
      nbrdig := 0 ;
      while (not eoln) do
        begin
          read(car) ;
          if ((ord(car) >= 65) and (ord(car) <= 70))
            then
              begin
                k := k + 1 ;
                tabin[k] := ord(car) - 44 ;
                nbrdig := 0
              end
            else
              if ((ord(car) >= 48) and (ord(car) <= 57))
                then
                  begin
                    if (nbrdig = 0)
                      then
                        begin
                          k := k + 1 ;
                          tabin[k] := (ord(car) - 48) + 30
                        end
                      else
                        tabin[k]:=((ord(car)-48)+((tabin[k]-30)*10))+30 ;
                    nbrdig := nbrdig + 1 ;
                  end
                else
                  if (ord(car) <> 32)
                    then
                      begin
                        case ord(car) of
                          47 : begin
                                 k := k + 1 ;
                                 tabin[k] := 1 ;
                               end ;
                          43 : begin
                                 k := k + 1 ;
                                 tabin[k] := 2 ;
                               end ;
                          42 : begin
                                 k := k + 1 ;
                                 tabin[k] := 4 ;
                               end ;
                          58 : begin
                                 k := k + 1 ;
                                 tabin[k] := 5 ;
                               end ;
                          40 : begin
                                 k := k + 1 ;
                                 tabin[k] := 7 ;
                               end ;
                          41 : begin
                                 k := k + 1 ;
                                 tabin[k] := 9 ;
```

```
                        end ;
                    45 : begin
                           k := k + 1 ;
                           if (k=1)
                             then tabin[k] := 8
                             else tabin[k] := 3 ;
                         end ;
                    91 : begin
                           k := k + 1 ;
                           tabin[k] := 10 ;
                         end ;
                    93 : begin
                           k := k + 1 ;
                           tabin[k] := 11 ;
                         end ;
                  end ;
                nbrdig := 0
              end
            else nbrdig := 0 ;
    end ;
end ;



procedure INFPRE(entree:tab1;var sortie:tab1) ;

    var i,bi,bs,pile,indice,k,priorite : integer ;
        ok : boolean ;
        bornes : array[1..6,1..2] of integer ;

    procedure INDSUIV(bi,bs:integer;var prior:integer;var ind:integer) ;

        var priorite,unite,i : integer ;
            parenth : boolean ;

        begin
          prior := 0 ;
          unite := 0 ;
          parenth := false ;
          for i:=bi to bs do
            begin
              priorite := 0 ;
              if (entree[i] <= 20)
                then
                  begin
                    case entree[i] of
                      2,3,4,5 : if (not parenth) then priorite := 6 ;
                      7,10    : begin
                                  parenth := true ;
                                  priorite := 2 ;
                                end ;
                      8       : if (not parenth) then priorite := 5 ;
                      1       : if (not parenth) then priorite := 3 ;
                      9,11    : parenth := false ;
                              end ;
                    unite := 0
                  end
                else
                  if (not parenth)
                    then
```

```
                    begin
                      priorite := 1 ;
                      unite := unite + 1 ;
                      if ((unite = 2) and (not parenth))
                        then priorite := 4 ;
                    end ;
            if (priorite > prior)
              then
                begin
                  prior := priorite ;
                  ind := i ;
                end ;
          end;
      end ;

begin
  for i:=1 to max1 do sortie[i] := 0 ;
  ok := false ;
  i := 1 ;
  while ((i <= max1) and (not ok)) do
    if (entree[i] = 0)
      then
        ok := true
      else
        i := i + 1 ;
  bi := 1 ;
  bs := i - 1 ;
  pile := 1 ;
  k := 1 ;
  while (pile <> 0) do
    begin
      while (bi <= bs) do
        begin
          indsuiv(bi,bs,priorite,indice) ;
          case priorite of
            4       : begin
                        sortie[k] := 6 ;
                        k := k + 1 ;
                        bornes[pile,1] := indice ;
                        bornes[pile,2] := bs ;
                        pile := pile + 1 ;
                        bs := indice - 1
                      end ;
            2       : begin
                        if (entree[indice] = 7)
                          then sortie[k] := 7
                          else sortie[k] := 10 ;
                        k := k + 1 ;
                        bornes[pile,1] := indice + 1 ;
                        bornes[pile,2] := bs ;
                        pile := pile + 1 ;
                        bs := indice - 1
                      end ;
            1,3,5,6 : begin
                        sortie[k] := entree[indice] ;
                        k := k + 1 ;
                        bornes[pile,1] := indice + 1 ;
                        bornes[pile,2] := bs ;
                        pile := pile + 1 ;
                        bs := indice - 1
                      end ;
            0       : bi := bi + 1 ;
```

```
                end ;
              end ;
            pile := pile - 1 ;
            if (pile <> 0) then begin bi := bornes[pile,1] ;
                                      bs := bornes[pile,2] ;
                               end ;
        end ;
    end ;



procedure STOREGLE(pteur:integer;entree:tab1) ;

    var i : integer ;

    procedure FILSTOCK(var fils:tab4;var i:integer;curtab:integer) ;

        var ok,arit1 : boolean ;
            k  : integer ;

        function EXISTE(indice:integer;tabpteur:integer) : boolean ;

            var genre : integer ;
                ok    : boolean ;

            begin
              existe := false ;
              if (entree[indice] <> 0)
                then
                  begin
                    if (entree[indice] > 20)
                      then
                        if (entree[indice] >= 30) then genre:=30
                                                  else genre:=20
                      else
                        genre := entree[indice] ;
                    if (genre = typ[tabpteur])
                      then
                        if ((genre = 20)and(fg[tabpteur]=entree[indice]-20))
                          then
                            existe := true
                          else
                            if ((genre = 30) and
                               (fg[tabpteur] = entree[indice] - 30))
                              then
                                existe := true
                              else
                                  if (genre < 20)
                                    then
                                      if (existe(indice + 1,fg[tabpteur]))
                                        then
                                          if (fd[tabpteur] = 0)
                                            then existe:=true
                                            else
                                              begin
                                      ok := existe(indice+2,fd[tabpteur]) ;
                                                existe := ok ;
                                              end ;
                  end ;
            end ;
```

```
    begin
      i := i + 1 ;
      ok := false ;
      arit1 := false ;
      k := 1 ;
      while ((k <= maxim) and (not ok)) do
        if (k <> curtab)
          then
            if (existe(i,k)) then ok := true
                             else k := k + 1
          else k:=k + 1 ;
        if (ok)
          then
            begin
              fils[curtab] := k ;
              if (entree[i] <= 20) then i := i + 2
            end
          else
            begin
              fils[curtab] := maxim + 1 ;
              if (entree[i] > 20)
                then
                  if (entree[i] >= 30)
                    then
                      begin
                        typ[maxim + 1] := 30 ;
                        fg[maxim + 1] := entree[i] - 30 ;
                        maxim := maxim + 1 ;
                        curtab := maxim
                      end
                    else
                      begin
                        typ[maxim + 1] := 20 ;
                        fg[maxim + 1] := entree[i] - 20 ;
                        maxim := maxim + 1 ;
                        curtab := maxim
                      end
                else
                  begin
                    typ[maxim + 1] := entree[i] ;
                    maxim := maxim + 1 ;
                    curtab := maxim ;
                    if (entree[i] = 8) or (entree[i] = 7) or
                       (entree[i] = 10)
                      then arit1 := true ;
                    filstock(fg,i,curtab) ;
                    if (not arit1) then filstock(fd,i,curtab) ;
                  end ;
            end ;
    end ;

begin
  if (entree[1] <= 20)
    then
      begin
        if (entree[1] <> 0)
          then
            begin
              typ[pteur] := entree[1] ;
              i := 1 ;
              filstock(fg,i,pteur) ;
```

```
                if (entree[i+1] <> 0) then filstock(fd,i,pteur) ;
              end ;
        end
      else
        if (entree[1] >= 30)
          then
            begin
              typ[pteur] := 30 ;
              fg[pteur] := entree[1] - 30
            end
          else
            begin
              typ[pteur] := 20 ;
              fg[pteur] := entree[1] - 20 ;
            end ;
  end ;



procedure PREINF ;

  var i,k,bi,bs,indicep : integer ;
      bornes : array[1..6,1..3] of integer ;
      parenth : boolean ;
      posouv,posfer : integer ;
      sortie : array[1..max3] of integer ;

  begin
    for i:=1 to max3 do sortie[i]:=0 ;
    i := expr ;
    bi := 1 ;
    bs := max3 ;
    parenth := false ;
    indicep := 1 ;
    while (indicep <> 0) do
      if (typ[i] <= 20) then
        begin
          if (typ[i] = 7) or (typ[i] = 10)
            then
              begin
                sortie[(bi+bs) div 2] := 9 ;
                posfer := (bi + bs) div 2 ;
                posouv := posfer ;
                parenth := true ;
              end
            else
              begin
                if (parenth) and (((bi+bs)div 2) < posouv)
                  then posouv := (bi+bs)div 2 ;
                sortie[(bi+bs) div 2] := typ[i] ;
                if (fd[i] <> 0) then
                  begin
                    bornes[indicep,1] := fd[i] ;
                    bornes[indicep,2] := ((bi + bs) div 2) + 1 ;
                    bornes[indicep,3] := bs ;
                    indicep := indicep + 1 ;
                  end ;
              end ;
            bs := ((bi + bs) div 2) - 1 ;
            i := fg[i]
```

```
      end
                        else
      begin
        if (parenth) and (((bi+bs)div 2) < posouv)
          then posouv := (bi+bs) div 2 ;
        if (typ[i] = 30) then sortie[(bi + bs) div 2] := fg[i] + 30
                         else sortie[(bi + bs) div 2] := fg[i] + 20 ;
        indicep := indicep - 1 ;
        if (indicep > 0) then
          begin
            if (parenth) and (bornes[indicep,3] > posfer)
              then
                begin
                  sortie[posouv - 1] := 7 ;
                  parenth := false ;
                end ;
              i := bornes[indicep,1] ;
              bi := bornes[indicep,2] ;
              bs := bornes[indicep,3]
          end
                          else
          if (parenth) then sortie[posouv-1] := 7 ;
      end ;
  k := 0 ;
  for i:=1 to max3 do tabout[i]:=0 ;
  for i:=1 to max3 do
    if (sortie[i] <> 0)
      then
        begin
          k := k + 1 ;
          tabout[k] := sortie[i] ;
        end ;
end ;



procedure OUTREGLE ;

 var i,k,j,cpteur1,cpteur2,step,ecart : integer ;
     slash,trouve : boolean ;
     prepa : array[1..max3] of integer ;
     tabcoord : array[1..max3,1..2] of integer ;

 procedure BIDON ;

     begin
       if (sortie[i] = 7)
         then
           begin
             cpteur2 := 0 ;
             trouve := false ;
             while (not trouve) do
               begin
                 trouve := (sortie[i] = 9) ;
                 prepa[k] := sortie[i] ;
                 coord1 := coord1 + 16 ;
                 tabcoord[k,1] := coord1 ;
                 tabcoord[k,2] := coord2 - 8 ;
                 k := k + 1 ;
                 i := i + 1 ;
```

```
            cpteur2 := cpteur2 + 1 ;
          end ;
        i := i - 1 ;
      end
    else
      begin
        cpteur2 := 1 ;
        prepa[k] := sortie[i] ;
        coord1 := coord1 + 16 ;
        tabcoord[k,1] := coord1 ;
        tabcoord[k,2] := coord2 - 8 ;
        k := k + 1 ;
      end ;
  slash := false ;
  coord1 := coord1 - 16 ;
  if (cpteur1 >= cpteur2)
    then
      begin
        ecart := (cpteur1 - cpteur2) div 2 ;
        for j:=1 to cpteur2 do
          tabcoord[k-j,1]:=tabcoord[k-j,1] -
                          ((cpteur2 + ecart) * 16) ;
        step := tabcoord[k-cpteur2-1,1] ;
        j := 0 ;
        while (j < ((cpteur1 * 2) - 1)) do
          begin
            prepa[k] := 1 ;
            tabcoord[k,1] := step - (j * 8) ;
            tabcoord[k,2] := coord2 ;
            k := k + 1 ;
            j := j + 1
          end
      end
    else
      begin
        for j:=1 to cpteur2 do
          tabcoord[k-j,1] :=
            tabcoord[k-j,1] - (cpteur1 * 16) ;
        ecart := (cpteur2 - cpteur1) div 2 ;
        for j:=1 to cpteur1 do
          tabcoord[k-cpteur2-j,1] :=
            tabcoord[k-cpteur2-j,1] + (ecart * 16) ;
        j := 0 ;
        step := tabcoord[k-1,1] ;
        while (j < ((cpteur2 * 2) - 1)) do
          begin
            prepa[k] := 1 ;
            tabcoord[k,1] := step - (j * 8) ;
            tabcoord[k,2] := coord2 ;
            k := k + 1 ;
            j := j + 1 ;
          end ;
      end ;
end ;
```

```
procedure IMP ;

    var j,w,interm : integer ;
        chiffre : string ;

    begin
      grafmode ;
      for j:=1 to (k-1) do
        begin
          moveto(tabcoord[j,1],tabcoord[j,2]) ;
          pencolor(white) ;
          if (prepa[j] > 20)
            then
              if (prepa[j] >= 30)
                then
                  begin
                    interm := prepa[j] - 30 ;
                    str(interm,chiffre) ;
                    wstring(chiffre)
                  end
                else
                  case prepa[j] of
                    21 : wchar('A') ;
                    22 : wchar('B') ;
                    23 : wchar('C') ;
                    24 : wchar('D') ;
                    25 : wchar('E') ;
                    26 : wchar('F') ;
                   end
            else
              case prepa[j] of
                1   : wchar('&') ;
                2   : wchar('+') ;
                3,8 : wchar('-') ;
                4   : wchar('*') ;
                5   : wchar(':') ;
                7   : wchar('(') ;
                9   : wchar(')') ;
                6   : for w:=j to (k-1) do
                        tabcoord[w,1] := tabcoord[w,1] - 16 ;
               end ;
          pencolor(none) ;
        end ;
    end ;

begin
  k := 1 ;
  slash := false ;
  i := 0 ;
  while (sortie[i+1] <> 0) do
    begin
      i := i + 1 ;
      if (sortie[i] <> 1)
        then
          if (not slash)
            then
              begin
                prepa[k] := sortie[i] ;
                coord1 := coord1 + 16 ;
                tabcoord[k,1] := coord1 ;
                tabcoord[k,2] := coord2 ;
                k := k + 1
```

```
                end
              else
                bidon
          else
            begin
              slash := true ;
              if (prepa[k-1] = 9)
                then
                  begin
                    trouve := false ;
                    cpteur1 := 0 ;
                    j := k - 1 ;
                    while (not trouve) do
                      begin
                        cpteur1 := cpteur1 + 1 ;
                        trouve := (prepa[j] = 7) ;
                        tabcoord[j,2] := coord2 + 8 ;
                        j := j - 1
                      end
                  end
                else
                  begin
                    tabcoord[k-1,2] := coord2 + 8 ;
                    cpteur1 := 1 ;
                  end ;
            end ;
      end ;
    imp ;
  end ;



procedure SAUVREGLE ;

    var j : integer ;

    begin
      rewrite(regle,'#5:REGLES.DATA') ;
      for j:=1 to maxrule do
        begin
          regle^[1] := typ[j] ;
          regle^[2] := fg[j] ;
          regle^[3] := fd[j] ;
          put(regle) ;
        end ;
      close(regle,lock) ;
    end ;



procedure COPREGLE ;

    begin
      reset(regle,'#5:REGLES.DATA') ;
      j := 1 ;
      while(not eof(regle)) do
        begin
          typ[j] := regle^[1] ;
```

```
        fg[j] := regle^[2] ;
        fd[j] := regle^[3] ;
        get(regle) ;
        j := j + 1 ;
      end ;
    close(regle) ;
  end ;



procedure SAUVPRIOR ;

  var j : integer ;

  begin
    rewrite(priorite,'#5:RPRIOR.DATA') ;
    for j:=1 to max3 do
      begin
        priorite^[1] := prior[j,1] ;
        priorite^[2] := prior[j,2] ;
        put(priorite) ;
      end ;
    close(prioritei,lock) ;
  end ;



procedure COPPRIOR ;

  var j : integer ;

  begin
    j := 1 ;
    reset(priorite,'#5:RPRIOR.DATA') ;
    while (j <= 47) do
      begin
        prior[j,1] := priorite^[1] ;
        prior[j,2] := priorite^[2] ;
        get(priorite) ;
        j := j + 1 ;
      end ;
    close(priorite,lock) ;
    prior[j,1] := 1 ;
    prior[j,2] := 12 ;
    prior[j+1,1] := 32 ;
    prior[j+1,2] := 42 ;
    pr1 := prior[j,1] ;
    pr2 := prior[j,2] ;
    pr3 := prior[j+1,1] ;
    pr4 := prior[j+1,2] ;
    maxpr := j - 1 ;
  end ;
```

```
procedure SAUVTER ;

    var j : integer ;

    begin
      rewrite(terme,'#5:TERMES.DATA') ;
      for j:=1 to max3 do
        begin
          terme^ := nbter[j] ;
          put(terme) ;
        end ;
      close(terme,lock) ;
    end ;



procedure COPTER ;

    var j : integer ;

    begin
      j := 1 ;
      reset(terme,'#5:TERMES.DATA') ;
      while (not eof(terme)) do
        begin
          nbter[j] := terme^ ;
          get(terme) ;
          j := j + 1 ;
        end ;
      close(terme,lock) ;
    end ;



procedure INCDT(numreg:integer;var j:integer) ;

    var car : char ;
        fini,arg : boolean ;

    begin
      fini := false ;
      while (not fini) do
        begin
          writeln('DONNEZ-MOI LA CONDITION') ;
          arg := false ;
          while (not eoln) do
            begin
              read(car) ;
              if ((ord(car) >= 65) and (ord(car) <= 70))
                then
                  if (not arg)
                    then
                      begin
                        arg := true ;
                        rcond[j,2] := ord(car) - 64
                      end
                    else
                      rcond[j,3] := ord(car) - 64
```

```
                else
                  if ((ord(car) >= 48) and (ord(car) <= 57))
                    then
                      if (not arg)
                        then
                          begin
                            arg := true ;
                            rcond[j,2] := (ord(car) - 48) + 10
                          end
                        else
                          rcond[j,3] := (ord(car) - 48) + 10
                    else
                      if (ord(car) <> 32)
                        then
                          case ord(car) of
                            62 : rcond[j,4] := 1 ;
                            60 : rcond[j,4] := 2 ;
                            38 : rcond[j,4] := 3 ;
                            94 : rcond[j,4] := 4 ;
                            47 : rcond[j,4] := 5 ;
                            33 : rcond[j,4] := 6 ;
                            63 : rcond[j,4] := 7 ;
                           end ;
              end ;
            rcond[j,1] := numreg ;
            writeln('Y A T-IL UNE AUTRE CONDITION ? ') ;
            readln(car) ;
            if (car = 'N') then fini:=true ;
            j := j + 1 ;
          end ;
      end ;
```

```
procedure SAUVCDTS ;

    var j : integer ;

    begin
      rewrite(cdts,'#5:CONDIT.DATA') ;
      for j:=1 to max1 do
        begin
          cdts^[1] := rcond[j,1] ;
          cdts^[2] := rcond[j,2] ;
          cdts^[3] := rcond[j,3] ;
          cdts^[4] := rcond[j,4] ;
          put(cdts) ;
        end ;
      close(cdts,lock) ;
    end ;
```

```
procedure COPCDTS ;

    begin
      j := 1 ;
      reset(cdts,'#5:CONDIT.DATA') ;
      while (not eof(cdts)) do
        begin
          rcond[j,1] := cdts^[1] ;
          rcond[j,2] := cdts^[2] ;
          rcond[j,3] := cdts^[3] ;
          rcond[j,4] := cdts^[4] ;
          get(cdts) ;
          j := j + 1 ;
        end ;
      close(cdts,lock) ;
    end ;
```

ANNEXE 4

```
const maxrule = 20 ;

type instant = record
          vari : char ;
          val  : integer ;
        end ;
     liste = array[1..6] of instant ;


var csymb : packed array[1..maxrule] of char ;
    symb : packed array[1..maxrule] of char ;
    arite:  array[1..maxrule] of integer ;
    arg  :  array[1..maxrule,1..2] of integer ;
    variable : set of char ;
    isymb : array[10..maxrule] of integer ;
    e : liste ;
    retour : boolean ;
    i,j,k : integer ;
    rep : char ;

  function UNIFIER(e1,e2:integer;var j:integer;var env:liste):boolean ;

      var present,ok : boolean ;
          i,indice   : integer ;

      begin
        if (symb[e1] in variable)
          then
            begin
              present := false ;
              for i:=1 to j do
                if (symb[e1] = env[i].vari) then
                                              begin
                                                present := true ;
                                                indice := i ;
                                                i := 7 ;
                                              end ;
              if (present)
                then
                  if (isymb[e2] = env[indice].val)
                    then unifier := true
                    else unifier := false
                else
                  begin
                    env[j+1].vari := symb[e1] ;
                    env[j+1].val := isymb[e2] ;
                    j := j + 1 ;
                    unifier := true
                  end
        end
      else
        if (symb[e1] = '0') or (symb[e1] = '1')
          then
            if ((isymb[2]=0)or(isymb[e2]=1)) then unifier := true
                                             else unifier := false
          else
            if ((symb[e1] <> csymb[e2]) or (arite[e1] <> arite[e2]))
              then
                unifier := false
              else
                begin
                  ok := true ;
```

```
                i := 1 ;
                while (ok) and (i <= arite[e1]) do
                  begin
                    ok := (unifier(arg[e1,i],arg[e2,i],j,env) and (ok)) ;
                    i := i + 1 ;
                  end ;
                unifier := ok ;
              end ;
end ;
```

```
const maxrule = 20 ;

type  liste = array[1..6] of integer ;

var   typ : array[1..maxrule] of integer ;
      fg  : array[1..maxrule] of integer ;
      fd  : array[1..maxrule] of integer ;
      env : liste ;
      retour : boolean ;
      j,i : integer ;
      rep : char ;

function UNIFIER(e1,e2:integer;var env:liste):boolean ;

    var ok : boolean ;

    begin
      if (typ[e1] >= 10) then
        if (typ[e1] = 10)  then
          if (env[fg[e1]] <> -1) then
            if (env[fg[e1]] = fg[e2]) then unifier:=true
                                      else unifier:=false
                                 else
            begin
              env[fg[e1]] := fg[e2] ;
              unifier := true
            end
                           else
          if (fg[e2] = fg[e1]) then unifier:=true
                               else unifier:=false
                         else
        if (typ[e1] <> typ[e2]) then unifier:=false
                                else
          begin
            ok := unifier(fg[e1],fg[e2],env) ;
            if ((ok) and (fd[e1] <> 0))
              then ok := ((ok) and (unifier(fd[e1],fd[e2],env)));
            unifier := ok ;
          end ;
    end ;
```

```
type tab = array[1..20] of integer ;
     liste = array[1..20] of integer ;

var j,k     : integer ;
    e1,e2 : tab ;
    env   : liste ;
    retour : boolean ;
    car    : char ;

function UNIFIER(var e1,e2:tab;var env:liste):boolean ;

    var i : integer ;
        ok : boolean ;

    begin
      i := 1 ;
      ok := true ;
      while ((i <= 20) and (e1[i] <> -1) and (ok)) do
        begin
          if (e1[i] <> e2[i]) then
            if (e1[i] <= 7) then ok:=false
                            else if (env[e1[i]] = -1)
                                 then env[e1[i]] := env[e2[i]]
                                 else if (env[e1[i]] <> env[e2[i]])
                                      then ok:=false ;
          i := i + 1 ;
        end ;
      unifier := ok ;
    end ;
```

```
const maxrule = 20 ;

type  liste = array[1..6] of integer ;

var typ : array[1..maxrule] of integer ;
    fg  : array[1..maxrule] of integer ;
    fd  : array[1..maxrule] of integer ;
    env : liste ;
    retour : boolean ;
    j,i : integer ;
    rep : char ;


function UNIFIER(e1,e2:integer;var env:liste):boolean ;

    var indicep : integer ;
        bornes  : array[1..6,1..2] of integer ;
        rule,expr : integer ;
        ok : boolean ;

    begin
      ok := true ;
      indicep := 1 ;
      rule := e1 ;
      expr := e2 ;
      while ((ok) and (indicep <> 0)) do
        if (typ[rule] >= 10) then
          if (typ[rule] = 10)  then
            if (env[fg[rule]] <> -1) then
              if (env[fg[rule]] <> fg[expr]) then ok:=false
                                             else
                if (indicep > 1) then begin indicep:=indicep - 1 ;
                                            rule := bornes[indicep,1] ;
                                            expr := bornes[indicep,2]
                                      end
                                 else indicep:=0
                                    else
              begin
                env[fg[rule]] := fg[expr] ;
                if (indicep > 1) then begin indicep:=indicep - 1 ;
                                            rule := bornes[indicep,1] ;
                                            expr := bornes[indicep,2]
                                      end
                                 else indicep:=0
              end
                               else
            if (fg[rule] <> fg[expr]) then ok:=false
                                      else if (indicep > 1) then
                                             begin
                                               indicep:=indicep - 1;
                                               rule:=bornes[indicep,1];
                                               expr:=bornes[indicep,2]
                                             end
                                                   else
                                             indicep:=0
                             else
          if(typ[rule] <> typ[expr]) then ok:=false
                                     else begin
                                            if (fd[rule] <> 0) then
                                              begin
                                                bornes[indicep,1]:=fd[rule];
                                                bornes[indicep,2]:=fd[expr];
```

```
                indicep := indicep + 1 ;
              end ;
            rule := fg[rule] ;
            expr := fg[expr] ;
          end ;
    unifier:=ok ;
  end ;
```

```
(*$G+*)
(*$R-*)

const maxrule = 20 ;

var typ : array[1..maxrule] of integer ;
    fg  : array[1..maxrule] of integer ;
    fd  : array[1..maxrule] of integer ;
    env : array[1..6] of integer ;
    retour : boolean ;
    j,i : integer ;
    rep : char ;

function UNIF(e1,e2:integer) : boolean ;

    label 5 ;

    begin
  5 : if (typ[e1] >= 20) and (typ[e2] = 30)
        then
         if (typ[e1] = 20)
          then
           if (env[fg[e1]] = -1)
            then
             begin
               env[fg[e1]]:=fg[e2] ;
               unif:=true
             end
            else
             unif:=(env[fg[e1]] = fg[e2])
          else
           unif:=(fg[e1] = fg[e2])
        else
         if (typ[e1] = typ[e2])
          then
           if (fd[e1] <> 0)
            then
               if (unif(fd[e1],fd[e2]))
                 then begin
                        e1:=fg[e1] ;
                        e2:=fg[e2] ;
                        goto 5
                      end
                 else unif:=false
            else
             begin
               e1:=fg[e1] ;
               e2:=fg[e2] ;
               goto 5
             end
          else
           unif:=false ;
    end ;
```

ANNEXE 5

```
procedure PROBL(ptdeb:integer;var etab:integer) ;

    begin
      typ[ptdeb] := diff8 ;
      fg[ptdeb] := etab ;
      if (nat1 <> 0)
        then
          if (num1 <> 0)
            then
              begin
                typ[etab] := 6 ;
                fg[etab] := etab + 1 ;
                typ[etab + 1] := 30 ;
                fg[etab + 1] := nat1 ;
                fd[etab] := etab + 2 ;
                etab := etab + 2 ;
                typ[etab] := 1 ;
                fg[etab] := etab + 1 ;
                fd[etab] := etab + 2 ;
                etab := etab + 1 ;
                typ[etab] := 30 ;
                fg[etab] := num1 ;
                etab := etab + 1 ;
                typ[etab] := 30 ;
                fg[etab] := denom1 ;
                etab := etab + 1
              end
            else
              begin
                typ[etab] := 30 ;
                fg[etab] := nat1 ;
                etab := etab + 1
              end
        else
          begin
            typ[etab] := 1 ;
            fg[etab] := etab + 1 ;
            fd[etab] := etab + 2 ;
            etab := etab + 1 ;
            typ[etab] := 30 ;
            fg[etab] := num1 ;
            etab := etab + 1 ;
            typ[etab] := 30 ;
            fg[etab] := denom1 ;
            etab := etab + 1
          end ;
      fd[ptdeb] := etab ;
      if (nat2 <> 0)
        then
          if (num2 <> 0)
            then
              begin
                typ[etab] := 6 ;
                fg[etab] := etab + 1 ;
                typ[etab + 1] := 30 ;
                fg[etab + 1] := nat2 ;
                fd[etab] := etab + 2 ;
                etab := etab + 2 ;
                typ[etab] := 1 ;
                fg[etab] := etab + 1 ;
                fd[etab] := etab + 2 ;
                etab := etab + 1 ;
```

```
                typ[etab] := 30 ;
                fg[etab] := num2 ;
                etab := etab + 1 ;
                typ[etab] := 30 ;
                fg[etab] := denom2 ;
                etab := etab + 1
              end
            else
              begin
                typ[etab] := 30 ;
                fg[etab] := nat2 ;
                etab := etab + 1
              end
        else
          begin
            typ[etab] := 1 ;
            fg[etab] := etab + 1 ;
            fd[etab] := etab + 2 ;
            etab := etab + 1 ;
            typ[etab] := 30 ;
            fg[etab] := num2 ;
            etab := etab + 1 ;
            typ[etab] := 30 ;
            fg[etab] := denom2 ;
            etab := etab + 1
          end ;
    end ;
```

```
procedure PEUNIF(e1,e2:integer;var i:integer;var trouve:boolean) ;

        var expn,z : integer ;

        procedure PE2UNIF ;

            begin
              fd[essai] := essai + 1 ;
              while (i < (nbtee[e2] - nbter[e1])) and (not trouve) do
                begin
                  typ[essai] := typ[expn] ;
                  fg[essai] := fg[expn] ;
                  typ[essai + 1] := typ[fd[expn]] ;
                  fg[essai + 1] := fg[fd[expn]] ;
                  fd[essai + 1] := fg[fd[fd[expn]]] ;
                  for z:=1 to maxvar do env[z] := -1 ;
                  if (unif(e1,essai)) then verifcdt(e1,e2,trouve) ;
                  i := i + 1 ;
                  expn := fd[expn] ;
                end ;
              if (not trouve)
                then
                  begin
                    typ[essai] := typ[expn] ;
                    fg[essai] := fg[expn] ;
                    typ[essai + 1] := typ[fd[expn]] ;
                    fg[essai + 1] := fg[fd[expn]] ;
                    fd[essai + 1] := fd[fd[expn]] ;
                    for z:=1 to maxvar do env[z] := -1 ;
                    if (unif(e1,essai)) then verifcdt(e1,e2,trouve) ;
                    i := i + 1 ;
                  end ;
            end ;

        begin
          i := 0 ;
          expn := e2 ;
          trouve := false ;
          if (nbter[e1] = 1)
            then
              begin
                if (typ[fg[e2]] =7) then i:=i+1 ;
                if (typ[fd[e2]] =7) then i:=i+1 ;
                while (i < (nbtee[e2] - 1)) and (not trouve) do
                  begin
                    for z:=1 to maxvar do env[z] := -1 ;
                    if (unif(e1,fg[expn])) then verifcdt(e1,e2,trouve) ;
                    i := i + 1 ;
                    expn := fd[expn] ;
                  end ;
                if (not trouve)
                  then
                    begin
                      for z:=1 to maxvar do env[z] := -1 ;
                      if (unif(e1,expn)) then verifcdt(e1,e2,trouve) ;
                      i := i + 1 ;
                    end ;
              end
            else
              if (nbter[e1] = 2)
                then
                  begin
```

```
                if (typ[fg[e2]] =7) then i:=i+1 ;
                if (typ[fd[e2]] =7) then i:=i+1 ;
                while (i < (nbtee[e2] - nbter[e1])) and (not trouve) do
                  begin
                    typ[essai] := typ[expn] ;
                    fg[essai] := fg[expn] ;
                    fd[essai] := fg[fd[expn]] ;
                    for z:=1 to maxvar do env[z] := -1 ;
                    if (unif(e1,essai)) then verifcdt(e1,e2,trouve) ;
                    i := i + 1 ;
                    expn := fd[expn] ;
                  end ;
                if (not trouve)
                  then
                    begin
                      typ[essai] := typ[expn] ;
                      fg[essai] := fg[expn] ;
                      fd[essai] := fd[expn] ;
                      for z:=1 to maxvar do env[z] := -1 ;
                      if (unif(e1,essai)) then verifcdt(e1,e2,trouve) ;
                      i := i + 1 ;
                    end ;
              end
            else
              pe2unif ;
    end ;



procedure VERIFCDT ;

    var arg1,arg2,i,interm,reste : integer ;

    begin
      verif := true ;
      i := 1 ;
      while(i <= max1) and (reaunif >= rcond[i,1]) and (verif) do
        if (reaunif > rcond[i,1])
          then i:=i + 1
          else
            begin
              if (rcond[i,2] < 10)
                then arg1 := env[rcond[i,2]]
                else arg1 := rcond[i,2] - 10 ;
              if (rcond[i,3] < 10)
                then arg2 := env[rcond[i,3]]
                else arg2 := rcond[i,3] - 10 ;
              case rcond[i,4] of
                1 : verif := (arg1 > arg2) ;
                2 : verif := (arg1 < arg2) ;
                3 : verif := (arg1 >= arg2) ;
                4 : begin
                      if (arg1 < arg2)
                        then
                          begin
                            interm := arg1 ;
                            arg1 := arg2 ;
                            arg2 := interm ;
                          end ;
                      reste := (arg1 mod arg2) ;
```

```
                    while (reste <> 0) do
                      begin
                        arg1 := arg2 ;
                        arg2 := reste ;
                        reste := (arg1 mod arg2) ;
                      end ;
                    verif := (arg2 = 1) ;
                  end ;
              5 : verif := (arg1 mod arg2 = 0) ;
              6 : if (nbtee[expaunif] = 1)
                    then verif := false
                    else
                      begin
                        arg1 := env[1] ;
                        peunif(16,expaunif,interm,verif) ;
                        if ((verif) or (typ[fg[expaunif]]=7) or
                                       (typ[fd[expaunif]]=7) or
                           ((typ[expaunif]=2) and (nbtee[expaunif]=2)))
                          then
                            verif := false
                          else
                            begin
                              verif := true ;
                              env[1] := arg1 ;
                            end ;
                      end ;
              7 : begin
                    if (arg1 < arg2)
                      then
                        begin
                          interm := arg1 ;
                          arg1 := arg2 ;
                          arg2 := interm ;
                        end ;
                    reste := (arg1 mod arg2) ;
                    while (reste <> 0) do
                      begin
                        arg1 := arg2 ;
                        arg2 := reste ;
                        reste := (arg1 mod arg2) ;
                      end ;
                    verif := (arg2 > 1) ;
                    if (verif) then env[3] := arg2 ;
                  end ;
            end ;
          i := i + 1 ;
        end ;
    end ;



procedure DEDUIRE(rpteur:integer;var fils:tab4;var etab:integer;
                  ptdeb:integer) ;

    var op,arg1,arg2,nbre : integer ;

    begin
      if (typ[rpteur] <> 0)
        then
          if (typ[rpteur] < 20)
```

```
        then
          if (typ[rpteur] = 10)
            then
              begin
                op := typ[fg[rpteur]] ;
                arg1 := env[fg[fg[fg[rpteur]]]] ;
                arg2 := env[fg[fd[fg[rpteur]]]] ;
                case op of
                  2 : nbre := arg1 + arg2 ;
                  3 : nbre := arg1 - arg2 ;
                  4 : nbre := arg1 * arg2 ;
                  5 : nbre := arg1 div arg2 ;
                 end ;
                if (typ[ptdeb] = 0)
                  then
                    begin
                      typ[ptdeb] := 30 ;
                      fg[ptdeb] := nbre
                    end
                  else
                    begin
                      fils[ptdeb] := etab ;
                      typ[etab] := 30 ;
                      fg[etab] := nbre ;
                      etab := etab + 1
                    end
              end
            else
              if (typ[ptdeb] = 0)
                then
                  begin
                    typ[ptdeb] := typ[rpteur] ;
                    deduire(fg[rpteur],fg,etab,ptdeb) ;
                    if (fd[rpteur] <> 0) then
                      deduire(fd[rpteur],fd,etab,ptdeb)
                  end
                else
                  begin
                    fils[ptdeb] := etab ;
                    typ[etab] := typ[rpteur] ;
                    etab := etab + 1 ;
                    ptdeb := etab - 1 ;
                    deduire(fg[rpteur],fg,etab,ptdeb) ;
                    if (fd[rpteur] <> 0) then
                      deduire(fd[rpteur],fd,etab,ptdeb)
                  end
        else
          if (typ[ptdeb] = 0)
            then
              begin
                typ[ptdeb] := 30 ;
                fg[ptdeb] := env[fg[rpteur]]
              end
            else
              begin
                fils[ptdeb] := etab ;
                typ[etab] := 30 ;
                fg[etab] := env[fg[rpteur]] ;
                etab := etab + 1 ;
              end ;
end ;
```

```
procedure PEDEDUIRE(rpteur,expteur:integer;var etab:integer;
                    j,ptdeb:integer) ;

    var expn,z,maxe,ref1,ref2 : integer ;

    begin
      if (j=1)
        then
          begin
            if (nbter[rpteur] < 0)
              then
                begin
                  expn := expteur ;
                  z := 1 ;
                  while (z <= ((nbter[rpteur - 25]) - 1)) do
                    begin
                      expn := fd[expn] ;
                      z := z + 1 ;
                    end ;
                  typ[ptdeb] := typ[expn] ;
                  fd[ptdeb] := fd[expn] ;
                  deduire(rpteur,fg,etab,ptdeb) ;
                end ;
            if ((nbter[rpteur] = 0) or (typ[rpteur] = 7))
              then
                begin
                  typ[ptdeb] := typ[expteur] ;
                  fd[ptdeb] := fd[expteur] ;
                  deduire(rpteur,fg,etab,ptdeb) ;
                end ;
            if ((nbter[rpteur] > 0) and (typ[rpteur] <> 7))
              then
                begin
                  deduire(rpteur,fg,etab,ptdeb) ;
                  typ[etab] := typ[expteur] ;
                  fg[etab] := fd[ptdeb] ;
                  fd[etab] := fd[expteur] ;
                  fd[ptdeb] := etab ;
                  etab := etab + 1 ;
                end ;
          end
        else
          begin
            maxe := expteur ;
            typ[ptdeb] := typ[expteur] ;
            fg[ptdeb] := fg[expteur] ;
            expn := ptdeb ;
            z := 3 ;
            while ((z <= j) and (typ[rpteur] <> 7)) do
              begin
                fd[expn] := etab ;
                typ[etab] := typ[fd[expteur]] ;
                fg[etab] := fg[fd[expteur]] ;
                expn := etab ;
                expteur := fd[expteur] ;
                etab := etab + 1 ;
                z := z + 1 ;
              end ;
```

```
            deduire(rpteur,fd,etab,expn) ;
            if ((nbter[rpteur] >= 0) and (j = nbtee[maxe])) or
    ((nbter[rpteur] <= 0) and (j+nbter[rpteur-25]-1 = nbtee[maxe]))
              then
                begin
                end
              else
                begin
                  ref1 := expn ;
                  ref2 := ref1 ;
                  while (typ[ref1] <> 1) and (typ[ref1] <> 30) do
                    begin
                      ref2 := ref1 ;
                      ref1 := fd[ref1] ;
                    end ;
                  fg[etab] := fd[ref2] ;
                  fd[ref2] := etab ;
                  for z:=1 to (nbter[rpteur -25]) do
                    expteur := fd[expteur] ;
                  typ[etab] := typ[expteur] ;
                  fd[etab] := fd[expteur] ;
                  etab := etab + 1 ;
                end ;
            if (rpteur - 25) = 23 then
              begin
                typ[maxe+1] := 3 ;
                fg[fd[maxe+1]] := fg[fd[maxe+1]] + 1;
              end ;
          end ;
    end ;


procedure MOTEUR ;

    var nbtexpr,nbterme,i,j : integer ;
        fini : boolean ;

    begin
      nbtexpr := nbtee[expr] ;
      i := 0 ;
      nbterme := nbtexpr ;
      while (nbtexpr <> 0) do
        begin
          while ((nbtexpr - i) > 0) do
            begin
              j := minr ;
              while (prior[j,2] < (((nbtexpr - i)*10)+1)) and (j <= maxr)
                 do j := j + 1 ;
              while (nbtexpr >= nbterme) and
              (prior[j,2] = (((nbterme - i)* 10) + 1)) do
                begin
                  anter := prior[j,1] ;
                  for z:=1 to maxvar do env[z]:=-1 ;
                  if (nbtee[expr] = nbter[anter])
                    then
                      if (unif(anter,expr))
                        then
                          begin
                            verifcdt(anter,expr,retour) ;
```

```
                    if (retour)
                      then deduire(anter+25,fg,etab,ptdeb) ;
                  end ;
              else
                begin
                  peunif(anter,expr,indice,retour) ;
                  if (retour)
                    then
                      pededuire(anter+25,expr,etab,
                                indice,ptdeb) ;
                end ;
            if (retour)
              then
                begin
                  nbtee[ptdeb] := nbtee[expr] + nbter[anter + 25] ;
                  ptdeb := ptdeb + 1 ;
                  expr := expr + 1 ;
                  nbtexpr := nbtee[expr] ;
                  retour := false ;
                end
              else
                j := j + 1 ;
          end ;
        if (nbtexpr = nbterme)
          then
            i := i + 1
          else
            begin
              i := 0 ;
              nbterme := nbtexpr ;
            end ;
      end ;
    i := 0 ;
    fini := false ;
    while (nbtexpr = nbterme) and (not fini) do
      begin
        j := minr ;
        while (prior[j,2] < (nbtexpr * 10)) and (j <= maxr)
          do j := j + 1 ;
        while (nbtexpr = nbterme) and
          (prior[j,2] = (nbterme * 10)) do
          begin
            anter := prior[j,1] ;
            for z:=1 to maxvar do env[z]:=-1 ;
            if (unif(anter,expr))
              then
                begin
                  verifcdt(anter,expr,retour) ;
                  if (retour)
                    then deduire(anter+25,fg,etab,ptdeb) ;
                end ;
            if (retour)
              then
                begin
                  nbtee[ptdeb] := nbtee[expr] + nbter[anter + 25] ;
                  ptdeb := ptdeb + 1 ;
                  expr := expr + 1 ;
                  nbtexpr := nbtee[expr] ;
                  retour := false
                end
              else
                j := j + 1 ;
```

```
            end ;
          if (nbtexpr <> nbterme) then nbterme := nbtexpr
                                  else fini := true ;
        end ;
    end ;
end ;
```

ANNEXE 6

```
(*$S+*)
unit DECLAR ;

  interface

    const max1 = 20 ;
          max3 = 50 ;
          maxt = 80 ;
          maxrule = 200 ;

    type tab4   = array[1..maxrule] of integer;

    var  nbter  : array[1..max3] of integer ;
         rcond  : array[1..max1,1..4] of integer ;
         typ,fg,fd : tab4 ;
         prior  : array[1..max3,1..2] of integer ;
         i,z    : integer ;

    procedure TOTO ;


  implementation

    procedure TOTO ;
       begin
       end ;

  begin
  end.
```

```
(*$S+*)
unit EXPL ;

  interface

    uses turtlegraphics ,
         (*$U #5:DEC.CODE *) declar ;

    type tab3 = array[1..max3] of integer ;
         tabt = array[1..maxt] of integer ;

    procedure PREINF(expr:integer;var tabout:tab3) ;
    procedure OUTREGLE(coord1,coord2:integer;sortie:tab3) ;
    procedure DEMEXPL ;


  implementation

    var coord1,coord2 : integer ;
        tabinf : tab3 ;

    procedure PREINF ;

        var i,k,bi,bs,indicep : integer ;
            bornes : array[1..6,1..3] of integer ;
            parenth : boolean ;
            posouv,posfer : integer ;
            sortie : array[1..maxt] of integer ;

        begin
          for i:=1 to maxt do sortie[i]:=0 ;
          i := expr ;
          bi := 1 ;
          bs := maxt ;
          parenth := false ;
          indicep := 1 ;
          while (indicep <> 0) do
            if (typ[i] <= 20) then
              begin
                if (typ[i] = 7) or (typ[i] = 10)
                  then
                    begin
                      sortie[(bi+bs) div 2] := 9 ;
                      posfer := (bi + bs) div 2 ;
                      posouv := posfer ;
                      parenth := true ;
                    end
                  else
                    begin
                      if (parenth) and (((bi+bs)div 2) < posouv)
                        then posouv := (bi+bs)div 2 ;
                      sortie[(bi+bs) div 2] := typ[i] ;
                      if (fd[i] <> 0) then
                        begin
                          bornes[indicep,1] := fd[i] ;
                          bornes[indicep,2] := ((bi + bs) div 2) + 1 ;
                          bornes[indicep,3] := bs ;
                          indicep := indicep + 1 ;
                        end ;
                    end ;
                bs := ((bi + bs) div 2) - 1 ;
                i := fg[i]
```

```
          end
                          else
          begin
            if (parenth) and (((bi+bs)div 2) < posouv)
              then posouv := (bi+bs) div 2 ;
            if (typ[i] = 30) then sortie[(bi + bs) div 2] := fg[i] + 30
                             else sortie[(bi + bs) div 2] := fg[i] + 20 ;
            indicep := indicep - 1 ;
            if (indicep > 0) then
              begin
                if (parenth) and (bornes[indicep,3] > posfer)
                  then
                    begin
                      sortie[posouv - 1] := 7 ;
                      parenth := false ;
                    end ;
                i := bornes[indicep,1] ;
                bi := bornes[indicep,2] ;
                bs := bornes[indicep,3]
              end
                              else
              if (parenth) then sortie[posouv-1] := 7 ;
          end ;
        k := 0 ;
        for i:=1 to max3 do tabout[i]:=0 ;
        for i:=1 to maxt do
          if (sortie[i] <> 0)
            then
              begin
                k := k + 1 ;
                tabout[k] := sortie[i] ;
              end ;
      end ;

procedure OUTREGLE ;

      var i,k,j,cpteur1,cpteur2,step,ecart : integer ;
          slash,trouve : boolean ;
          prepa : array[1..max3] of integer ;
          tabcoord : array[1..max3,1..2] of integer ;

      procedure BIDON ;

          begin
            if (sortie[i] = 7)
              then
                begin
                  cpteur2 := 0 ;
                  trouve := false ;
                  while (not trouve) do
                    begin
                      trouve := (sortie[i] = 9) ;
                      prepa[k] := sortie[i] ;
                      coord1 := coord1 + 16 ;
                      tabcoord[k,1] := coord1 ;
                      tabcoord[k,2] := coord2 - 8 ;
                      k := k + 1 ;
                      i := i + 1 ;
                      cpteur2 := cpteur2 + 1 ;
                    end ;
                  i := i - 1 ;
```

```
            end
          else
            begin
              cpteur2 := 1 ;
              prepa[k] := sortie[i] ;
              coord1 := coord1 + 16 ;
              tabcoord[k,1] := coord1 ;
              tabcoord[k,2] := coord2 - 8 ;
              k := k + 1 ;
            end ;
        slash := false ;
        coord1 := coord1 - 16 ;
        if (cpteur1 >= cpteur2)
          then
            begin
              ecart := (cpteur1 - cpteur2) div 2 ;
              for j:=1 to cpteur2 do
                tabcoord[k-j,1]:=tabcoord[k-j,1] -
                                  ((cpteur2 + ecart) * 16) ;
              step := tabcoord[k-cpteur2-1,1] ;
              j := 0 ;
              while (j < ((cpteur1 * 2) - 1)) do
                begin
                  prepa[k] := 1 ;
                  tabcoord[k,1] := step - (j * 8) ;
                  tabcoord[k,2] := coord2 ;
                  k := k + 1 ;
                  j := j + 1
                end
            end
          else
            begin
              for j:=1 to cpteur2 do
                tabcoord[k-j,1] :=
                  tabcoord[k-j,1] - (cpteur1 * 16) ;
              ecart := (cpteur2 - cpteur1) div 2 ;
              for j:=1 to cpteur1 do
                tabcoord[k-cpteur2-j,1] :=
                  tabcoord[k-cpteur2-j,1] + (ecart * 16) ;
              j := 0 ;
              step := tabcoord[k-1,1] ;
              while (j < ((cpteur2 * 2) - 1)) do
                begin
                  prepa[k] := 1 ;
                  tabcoord[k,1] := step - (j * 8) ;
                  tabcoord[k,2] := coord2 ;
                  k := k + 1 ;
                  j := j + 1 ;
                end ;
            end ;
      end ;


procedure IMP ;

    var i,w,interm : integer ;
        chiffre : string ;

    begin
      grafmode ;
      for j:=1 to (k-1) do
        begin
```

```
            moveto(tabcoord[j,1],tabcoord[j,2]) ;
            pencolor(white) ;
            if (prepa[j] > 20)
              then
                if (prepa[j] >= 30)
                  then
                    begin
                      interm := prepa[j] - 30 ;
                      str(interm,chiffre) ;
                      wstring(chiffre)
                    end
                  else
                    case prepa[j] of
                      21 : wchar('A') ;
                      22 : wchar('B') ;
                      23 : wchar('C') ;
                      24 : wchar('D') ;
                      25 : wchar('E') ;
                      26 : wchar('F') ;
                     end
              else
                case prepa[j] of
                  1   : wchar('&') ;
                  2   : wchar('+') ;
                  3,8 : wchar('-') ;
                  4   : wchar('*') ;
                  5   : wchar(':') ;
                  7   : wchar('(') ;
                  9   : wchar(')') ;
                  6   : for w:=j to (k-1) do
                          tabcoord[w,1] := tabcoord[w,1] - 16 ;
                 end ;
            pencolor(none) ;
          end ;
      end ;

begin
  k := 1 ;
  slash := false ;
  i := 0 ;
  while (sortie[i+1] <> 0) do
    begin
      i := i + 1 ;
      if (sortie[i] <> 1)
        then
          if (not slash)
            then
              begin
                prepa[k] := sortie[i] ;
                coord1 := coord1 + 16 ;
                tabcoord[k,1] := coord1 ;
                tabcoord[k,2] := coord2 ;
                k := k + 1
              end
            else
              bidon
        else
          begin
            slash := true ;
            if (prepa[k-1] = 9)
              then
                begin
```

```
                        trouve := false ;
                        cpteur1 := 0 ;
                        j := k - 1 ;
                        while (not trouve) do
                          begin
                            cpteur1 := cpteur1 + 1 ;
                            trouve := (prepa[j] = 7) ;
                            tabcoord[j,2] := coord2 + 8 ;
                            j := j - 1
                          end
                      end
                    else
                      begin
                        tabcoord[k-1,2] := coord2 + 8 ;
                        cpteur1 := 1 ;
                      end ;
                end ;
          end ;
        imp ;
    end ;

procedure DEMEXPL ;

    begin
       (*$R TURTLEGRAPHICS*)
       initturtle ;
       coord1 := 5 ;
       coord2 := 175 ;
       preinf(100,tabinf) ;
       outregle(coord1,coord2,tabinf) ;
       coord1 := 100 ;
       coord2 := 205 ;
       z := 101 ;
       while (typ[z] <> 0) do
         begin
           preinf(z,tabinf) ;
           if z=107 then
             begin
               moveto(160,4) ;
               pencolor(white) ;
               wstring('TAPER RETURN') ;
               pencolor(none) ;
               readln ;
               initturtle ;
               grafmode ;
               coord2 := 175
             end
                   else
               coord2 := coord2 - 30 ;
           moveto(coord1,coord2) ;
           pencolor(white) ;
           wchar('=') ;
           pencolor(none) ;
           outregle(coord1,coord2,tabinf) ;
           z := z + 1 ;
         end ;
       readln ;
       textmode ;
     end ;

     begin
```

```
(*$S+*)
unit FICHIER ;

  interface

    uses (*$U #5:DEC.CODE *) declar ;

    type array1 = file of integer ;
         array2 = file of array[1..2] of integer ;
         array3 = file of array[1..3] of integer ;
         array4 = file of array[1..4] of integer ;

    var  priorite:array2 ;
         regle  : array3 ;
         terme  : array1 ;
         cdts   : array4 ;
         pr1,pr2,pr3,pr4,maxpr : integer ;

    procedure SAUVPRIOR ;
    procedure COPPRIOR  ;
    procedure SAUVREGLE ;
    procedure COPREGLE(var j:integer) ;
    procedure SAUVTER   ;
    procedure COPTER    ;
    procedure SAUVCDTS  ;
    procedure COPCDTS   ;


  implementation

    procedure SAUVPRIOR ;

        var j : integer ;

        begin
          rewrite(priorite,'#5:RPRIOR.DATA') ;
          for j:=1 to max3 do
            begin
              priorite^[1] := prior[j,1] ;
              priorite^[2] := prior[j,2] ;
              put(priorite) ;
            end ;
          close(prioritei,lock) ;
        end ;


    procedure COPPRIOR ;

        var j : integer ;

        begin
          j := 1 ;
          reset(priorite,'#5:RPRIOR.DATA') ;
          while (j <= 47) do
            begin
              prior[j,1] := priorite^[1] ;
              prior[j,2] := priorite^[2] ;
              get(priorite) ;
              j := j + 1 ;
            end ;
          close(priorite,lock) ;
          prior[j,1] := 1 ;
```

```
      prior[j,2] := 12 ;
      prior[j+1,1] := 32 ;
      prior[j+1,2] := 42 ;
      pr1 := prior[j,1] ;
      pr2 := prior[j,2] ;
      pr3 := prior[j+1,1] ;
      pr4 := prior[j+1,2] ;
      maxpr := j - 1 ;
    end ;

procedure SAUVREGLE ;

    var j : integer ;

    begin
      rewrite(regle,'#5:REGLES.DATA') ;
      for j:=1 to maxrule do
        begin
          regle^[1] := typ[j] ;
          regle^[2] := fg[j] ;
          regle^[3] := fd[j] ;
          put(regle) ;
        end ;
      close(regle,lock) ;
    end ;

procedure COPREGLE ;

    begin
      reset(regle,'#5:REGLES.DATA') ;
      j := 1 ;
      while(not eof(regle)) do
        begin
          typ[j] := regle^[1] ;
          fg[j] := regle^[2] ;
          fd[j] := regle^[3] ;
          get(regle) ;
          j := j + 1 ;
        end ;
      close(regle) ;
    end ;

procedure SAUVTER ;

    var j : integer ;

    begin
      rewrite(terme,'#5:TERMES.DATA') ;
      for j:=1 to max3 do
        begin
          terme^ := nbter[j] ;
          put(terme) ;
        end ;
      close(terme,lock) ;
    end ;
```

```
procedure COPTER ;

    var j : integer ;

    begin
      j := 1 ;
      reset(terme,'#5:TERMES.DATA') ;
      while (not eof(terme)) do
        begin
          nbter[j] := terme^ ;
          get(terme) ;
          j := j + 1 ;
        end ;
      close(terme,lock) ;
    end ;


procedure SAUVCDTS ;

    var j : integer ;

    begin
      rewrite(cdts,'#5:CONDIT.DATA') ;
      for j:=1 to max1 do
        begin
          cdts^[1] := rcond[j,1] ;
          cdts^[2] := rcond[j,2] ;
          cdts^[3] := rcond[j,3] ;
          cdts^[4] := rcond[j,4] ;
          put(cdts) ;
        end ;
      close(cdts,lock) ;
    end ;


procedure COPCDTS ;

    begin
      j := 1 ;
      reset(cdts,'#5:CONDIT.DATA') ;
      while (not eof(cdts)) do
        begin
          rcond[j,1] := cdts^[1] ;
          rcond[j,2] := cdts^[2] ;
          rcond[j,3] := cdts^[3] ;
          rcond[j,4] := cdts^[4] ;
          get(cdts) ;
          j := j + 1 ;
        end ;
      close(cdts,lock) ;
    end ;


begin
end.
```

```
program SE ;

    (*$G+*)
    (*$S+*)
    (*$N+*)
    (*$R-*)

    uses applestuff,turtlegraphics,
         (*$U #5:DEC.CODE *) declar,
         (*$U #5:INOUT.CODE *) fichier,
         (*$U #5:IMPR.CODE *) expl ;

    const maxvar  = 6   ;
          max4    = 100 ;
          max5    = 120 ;

    type  tabl = array[1..8] of integer ;

    var env        : array[1..6] of integer ;
        nbtee      : array[max4..max5] of integer ;
        retour     : boolean ;
        anter      : integer ;
        consr,ptdeb,etab : integer ;
        essai      : integer ;
        expr       : integer ;
        indice     : integer ;
        diff7,diff8   : integer ;
        niv2          : integer ;
        diff2         : tabl    ;
        nat1,nat2     : integer ;
        denom1,denom2 : integer ;
        num1,num2     : integer ;
        minr,maxr     : integer ;
        rep         : char ;
        fin   : boolean ;

   (*$I #5:GENERER *)

    function UNIF(e1,e2:integer) : boolean ;

        label 5 ;

        begin
      5 : if (typ[e1] >= 20) and (typ[e2] = 30)
            then
             if (typ[e1] = 20)
              then
               if (env[fg[e1]] = -1)
                then
                 begin
                   env[fg[e1]]:=fg[e2] ;
                   unif:=true
                 end
                else
                 unif:=(env[fg[e1]] = fg[e2])
              else
               unif:=(fg[e1] = fg[e2])
            else
             if (typ[e1] = typ[e2])
              then
               if (fd[e1] <> 0)
                then
```

```
                if (unif(fd[e1],fd[e2]))
                  then begin
                         e1:=fg[e1] ;
                         e2:=fg[e2] ;
                         goto 5
                       end
                  else unif:=false
              else
               begin
                 e1:=fg[e1] ;
                 e2:=fg[e2] ;
                 goto 5
               end
            else
             unif:=false ;
        end ;

procedure VERIFCDT(reaunif,expaunif:integer;var verif:boolean) ; forward ;

procedure PEUNIF(e1,e2:integer;var i:integer;var trouve:boolean) ;

        var expn,z : integer ;

        procedure PE2UNIF ;

            begin
              fd[essai] := essai + 1 ;
              while (i < (nbtee[e2] - nbter[e1])) and (not trouve) do
                begin
                  typ[essai] := typ[expn] ;
                  fg[essai] := fg[expn] ;
                  typ[essai + 1] := typ[fd[expn]] ;
                  fg[essai + 1] := fg[fd[expn]] ;
                  fd[essai + 1] := fg[fd[fd[expn]]] ;
                  for z:=1 to maxvar do env[z] := -1 ;
                  if (unif(e1,essai)) then verifcdt(e1,e2,trouve) ;
                  i := i + 1 ;
                  expn := fd[expn] ;
                end ;
              if (not trouve)
                then
                  begin
                    typ[essai] := typ[expn] ;
                    fg[essai] := fg[expn] ;
                    typ[essai + 1] := typ[fd[expn]] ;
                    fg[essai + 1] := fg[fd[expn]] ;
                    fd[essai + 1] := fd[fd[expn]] ;
                    for z:=1 to maxvar do env[z] := -1 ;
                    if (unif(e1,essai)) then verifcdt(e1,e2,trouve) ;
                    i := i + 1 ;
                  end ;
            end ;

        begin
          i := 0 ;
          expn := e2 ;
          trouve := false ;
          if (nbter[e1] = 1)
            then
              begin
                if (typ[fg[e2]] =7) then i:=i+1 ;
```

```
            if (typ[fd[e2]] =7) then i:=i+1 ;
            while (i < (nbtee[e2] - 1)) and (not trouve) do
              begin
                for z:=1 to maxvar do env[z] := -1 ;
                if (unif(e1,fg[expn])) then verifcdt(e1,e2,trouve) ;
                i := i + 1 ;
                expn := fd[expn] ;
              end ;
            if (not trouve)
              then
                begin
                  for z:=1 to maxvar do env[z] := -1 ;
                  if (unif(e1,expn)) then verifcdt(e1,e2,trouve) ;
                  i := i + 1 ;
                end ;
          end
        else
          if (nbter[e1] = 2)
            then
              begin
                if (typ[fg[e2]] =7) then i:=i+1 ;
                if (typ[fd[e2]] =7) then i:=i+1 ;
                while (i < (nbtee[e2] - nbter[e1])) and (not trouve) do
                  begin
                    typ[essai] := typ[expn] ;
                    fg[essai] := fg[expn] ;
                    fd[essai] := fg[fd[expn]] ;
                    for z:=1 to maxvar do env[z] := -1 ;
                    if (unif(e1,essai)) then verifcdt(e1,e2,trouve) ;
                    i := i + 1 ;
                    expn := fd[expn] ;
                  end ;
                if (not trouve)
                  then
                    begin
                      typ[essai] := typ[expn] ;
                      fg[essai] := fg[expn] ;
                      fd[essai] := fd[expn] ;
                      for z:=1 to maxvar do env[z] := -1 ;
                      if (unif(e1,essai)) then verifcdt(e1,e2,trouve) ;
                      i := i + 1 ;
                    end ;
              end
            else
              pe2unif ;
    end ;

procedure VERIFCDT ;

    var arg1,arg2,i,interm,reste : integer ;

    begin
      verif := true ;
      i := 1 ;
      while(i <= max1) and (reaunif >= rcond[i,1]) and (verif) do
        if (reaunif > rcond[i,1])
          then i:=i + 1
          else
            begin
              if (rcond[i,2] < 10)
                then arg1 := env[rcond[i,2]]
                else arg1 := rcond[i,2] - 10 ;
```

```
if (rcond[i,3] < 10)
  then arg2 := env[rcond[i,3]]
  else arg2 := rcond[i,3] - 10 ;
case rcond[i,4] of
  1 : verif := (arg1 > arg2) ;
  2 : verif := (arg1 < arg2) ;
  3 : verif := (arg1 >= arg2) ;
  4 : begin
        if (arg1 < arg2)
          then
            begin
              interm := arg1 ;
              arg1 := arg2 ;
              arg2 := interm ;
            end ;
        reste := (arg1 mod arg2) ;
        while (reste <> 0) do
          begin
            arg1 := arg2 ;
            arg2 := reste ;
            reste := (arg1 mod arg2) ;
          end ;
        verif := (arg2 = 1) ;
      end ;
  5 : verif := (arg1 mod arg2 = 0) ;
  6 : if (nbtee[expaunif] = 1)
        then verif := false
        else
          begin
            arg1 := env[1] ;
            peunif(16,expaunif,interm,verif) ;
            if ((verif) or (typ[fg[expaunif]]=7) or
                           (typ[fd[expaunif]]=7) or
               ((typ[expaunif]=2) and (nbtee[expaunif]=2)))
              then
                verif := false
              else
                begin
                  verif := true ;
                  env[1] := arg1 ;
                end ;
          end ;
  7 : begin
        if (arg1 < arg2)
          then
            begin
              interm := arg1 ;
              arg1 := arg2 ;
              arg2 := interm ;
            end ;
        reste := (arg1 mod arg2) ;
        while (reste <> 0) do
          begin
            arg1 := arg2 ;
            arg2 := reste ;
            reste := (arg1 mod arg2) ;
          end ;
        verif := (arg2 > 1) ;
        if (verif) then env[3] := arg2 ;
      end ;
 end ;
i := i + 1 ;
```

```
            end ;
    end ;

procedure DEDUIRE(rpteur:integer;var fils:tab4;var etab:integer;
                  ptdeb:integer) ;

    var op,arg1,arg2,nbre : integer ;

    begin
      if (typ[rpteur] <> 0)
        then
          if (typ[rpteur] < 20)
            then
              if (typ[rpteur] = 10)
                then
                  begin
                    op := typ[fg[rpteur]] ;
                    arg1 := env[fg[fg[fg[rpteur]]]] ;
                    arg2 := env[fg[fd[fg[rpteur]]]] ;
                    case op of
                      2 : nbre := arg1 + arg2 ;
                      3 : nbre := arg1 - arg2 ;
                      4 : nbre := arg1 * arg2 ;
                      5 : nbre := arg1 div arg2 ;
                     end ;
                    if (typ[ptdeb] = 0)
                      then
                        begin
                          typ[ptdeb] := 30 ;
                          fg[ptdeb] := nbre
                        end
                      else
                        begin
                          fils[ptdeb] := etab ;
                          typ[etab] := 30 ;
                          fg[etab] := nbre ;
                          etab := etab + 1
                        end
                  end
                else
                  if (typ[ptdeb] = 0)
                    then
                      begin
                        typ[ptdeb] := typ[rpteur] ;
                        deduire(fg[rpteur],fg,etab,ptdeb) ;
                        if (fd[rpteur] <> 0) then
                          deduire(fd[rpteur],fd,etab,ptdeb)
                      end
                    else
                      begin
                        fils[ptdeb] := etab ;
                        typ[etab] := typ[rpteur] ;
                        etab := etab + 1 ;
                        ptdeb := etab - 1 ;
                        deduire(fg[rpteur],fg,etab,ptdeb) ;
                        if (fd[rpteur] <> 0) then
                          deduire(fd[rpteur],fd,etab,ptdeb)
                      end
            else
              if (typ[ptdeb] = 0)
                then
                  begin
```

```
                    typ[ptdeb] := 30 ;
                    fg[ptdeb] := env[fg[rpteur]]
                  end
                else
                  begin
                    fils[ptdeb] := etab ;
                    typ[etab] := 30 ;
                    fg[etab] := env[fg[rpteur]] ;
                    etab := etab + 1 ;
                  end ;
          end ;

procedure PEDEDUIRE(rpteur,expteur:integer;var etab:integer;
                    j,ptdeb:integer) ;

    var expn,z,maxe,ref1,ref2 : integer ;

    begin
      if (j=1)
        then
          begin
            if (nbter[rpteur] < 0)
              then
                begin
                  expn := expteur ;
                  z := 1 ;
                  while (z <= ((nbter[rpteur - 25]) - 1)) do
                    begin
                      expn := fd[expn] ;
                      z := z + 1 ;
                    end ;
                  typ[ptdeb] := typ[expn] ;
                  fd[ptdeb] := fd[expn] ;
                  deduire(rpteur,fg,etab,ptdeb) ;
                end ;
            if ((nbter[rpteur] = 0) or (typ[rpteur] = 7))
              then
                begin
                  typ[ptdeb] := typ[expteur] ;
                  fd[ptdeb] := fd[expteur] ;
                  deduire(rpteur,fg,etab,ptdeb) ;
                end ;
            if ((nbter[rpteur] > 0) and (typ[rpteur] <> 7))
              then
                begin
                  deduire(rpteur,fg,etab,ptdeb) ;
                  typ[etab] := typ[expteur] ;
                  fg[etab] := fd[ptdeb] ;
                  fd[etab] := fd[expteur] ;
                  fd[ptdeb] := etab ;
                  etab := etab + 1 ;
                end ;
          end
        else
          begin
            maxe := expteur ;
            typ[ptdeb] := typ[expteur] ;
            fg[ptdeb] := fg[expteur] ;
            expn := ptdeb ;
            z := 3 ;
            while ((z <= j) and (typ[rpteur] <> 7)) do
              begin
```

```
                fd[expn] := etab ;
                typ[etab] := typ[fd[expteur]] ;
                fg[etab] := fg[fd[expteur]] ;
                expn := etab ;
                expteur := fd[expteur] ;
                etab := etab + 1 ;
                z := z + 1 ;
              end ;
            deduire(rpteur,fd,etab,expn) ;
            if ((nbter[rpteur] >= 0) and (j = nbtee[maxe])) or
     ((nbter[rpteur] <= 0) and (j+nbter[rpteur-25]-1 = nbtee[maxe]))
              then
                begin
                end
              else
                begin
                  ref1 := expn ;
                  ref2 := ref1 ;
                  while (typ[ref1] <> 1) and (typ[ref1] <> 30) do
                    begin
                      ref2 := ref1 ;
                      ref1 := fd[ref1] ;
                    end ;
                  fg[etab] := fd[ref2] ;
                  fd[ref2] := etab ;
                  for z:=1 to (nbter[rpteur -25]) do
                    expteur := fd[expteur] ;
                  typ[etab] := typ[expteur] ;
                  fd[etab] := fd[expteur] ;
                  etab := etab + 1 ;
                end ;
            if (rpteur - 25) = 23 then
              begin
                typ[maxe+1] := 3 ;
                fg[fd[maxe+1]] := fg[fd[maxe+1]] + 1;
              end ;
          end ;
      end ;

procedure MOTEUR ;

    var nbtexpr,nbterme,i,j : integer ;
        fini : boolean ;

    begin
      nbtexpr := nbtee[expr] ;
      i := 0 ;
      nbterme := nbtexpr ;
      while (nbtexpr <> 0) do
        begin
          while ((nbtexpr - i) > 0) do
            begin
              j := minr ;
              while (prior[j,2] < (((nbtexpr - i)*10)+1)) and (j <= maxr)
                 do j := j + 1 ;
              while (nbtexpr >= nbterme) and
              (prior[j,2] = (((nbterme - i)* 10) + 1)) do
                begin
                  anter := prior[j,1] ;
                  for z:=1 to maxvar do env[z]:=-1 ;
                  if (nbtee[expr] = nbter[anter])
                    then
```

```
                if (unif(anter,expr))
                  then
                    begin
                      verifcdt(anter,expr,retour) ;
                      if (retour)
                        then deduire(anter+25,fg,etab,ptdeb) ;
                      end ;
                else
                  begin
                    peunif(anter,expr,indice,retour) ;
                    if (retour)
                      then
                        pededuire(anter+25,expr,etab,
                                  indice,ptdeb) ;
                  end ;
              if (retour)
                then
                  begin
                    nbtee[ptdeb] := nbtee[expr] + nbter[anter + 25] ;
                    ptdeb := ptdeb + 1 ;
                    expr := expr + 1 ;
                    nbtexpr := nbtee[expr] ;
                    retour := false ;
                  end
                else
                  j := j + 1 ;
            end ;
          if (nbtexpr = nbterme)
            then
              i := i + 1
            else
              begin
                i := 0 ;
                nbterme := nbtexpr ;
              end ;
        end ;
      i := 0 ;
      fini := false ;
      while (nbtexpr = nbterme) and (not fini) do
        begin
          j := minr ;
          while (prior[j,2] < (nbtexpr * 10)) and (j <= maxr)
            do j := j + 1 ;
          while (nbtexpr = nbterme) and
           (prior[j,2] = (nbterme * 10)) do
            begin
              anter := prior[j,1] ;
              for z:=1 to maxvar do env[z]:=-1 ;
              if (unif(anter,expr))
                then
                  begin
                    verifcdt(anter,expr,retour) ;
                    if (retour)
                      then deduire(anter+25,fg,etab,ptdeb) ;
                  end ;
              if (retour)
                then
                  begin
                    nbtee[ptdeb] := nbtee[expr] + nbter[anter + 25] ;
                    ptdeb := ptdeb + 1 ;
                    expr := expr + 1 ;
                    nbtexpr := nbtee[expr] ;
```

```
                        retour := false
                      end
                    else
                      j := j + 1 ;
                end ;
              if (nbtexpr <> nbterme) then nbterme := nbtexpr
                                      else fini := true ;
            end ;
        end ;
    end ;

  begin
    copter ;
    copcdts ;
    copregle(j) ;
    copprior ;
    j := 96 ;

    fin := false ;
   while (not fin) do
    begin
    for z:=j to maxrule do
      begin
        typ[z] := 0 ;
        fg[z] := 0 ;
        fd[z] := 0 ;
      end ;
    diff2[1] := 2 ;
    diff2[2] := 4 ;
    diff2[3] := 3 ;
    diff2[4] := 6 ;
    diff2[5] := 8 ;
    diff2[6] := 5 ;
    diff2[7] := 10 ;
    diff2[8] := 12 ;
writeln ;
write('DIFF8 ?  -->') ;
readln(diff8) ;
writeln ;
    (*REPEAT DIFF7:=RAND(9,13) UNTIL (DIFF7 <> 10) ;
    NIV2  := 8 ;
    WRITELN(DIFF7) ;
    READLN ;
    FRACTGEN(DIFF7,DIFF8,NIV2,DIFF2,NAT1,NAT2,DENOM1,DENOM2,NUM1,NUM2) ;
    PAGE(OUTPUT) ;
    WRITELN(NAT1,'  ',NUM1,' / ',DENOM1) ;
    WRITELN ;
    WRITELN(NAT2,'  ',NUM2,' / ',DENOM2) ;*)
    readln(nat1) ;
    readln(nat2) ;
    readln(denom1) ;
    readln(denom2) ;
    readln(num1) ;
    readln(num2) ;
    case diff8 of
      2 : begin
            minr := pr1 ;
            maxr := pr2 - 1 ;
          end ;
      3 : begin
            minr := pr2 ;
            maxr := pr3 - 1 ;
```

```
            end ;
        4 : begin
              minr := pr3 ;
              maxr := pr4 - 1 ;
            end ;
        5 : begin
              minr := pr4 ;
              maxr := maxpr ;
            end ;
       end ;
      expr := 100 ;
      nbtee[expr] := 2 ;
      ptdeb := 100 ;
      etab := 121 ;
      probl(ptdeb,etab) ;
      ptdeb := ptdeb + 1 ;
      essai := maxrule - 1 ;
      moteur ;
      demexpl ;
      writeln('FINI ?') ;
      readln(rep) ;
      if (rep = 'Y') then fin:=true ;
     end ;
end.
```

ANNEXE 7

```
program TEST(input,output);

    (*$S+*)

    uses turtlegraphics,transcend ;

    var i,x2,x1,color   : integer ;
        y1,y2           : integer ;
        rep     : char ;

    procedure AVANCE(x1:integer;x2:integer;long:integer);external;

    procedure INIT ; external ;

    procedure MTEXT ; external ;

    procedure GRAF ; external ;

    procedure ALLER(x1,y1:integer;x2,y2:integer) ;

        var x10,y10,x20,y20 : integer ;
            r,cosinus,invcos,sinus,r10 : real ;

        begin
          if (x1 <> x2) and (y1 <> y2) then
            begin
              x10 := x2 - x1 ;
              y10 := y2 - y1 ;
              r := sqrt((x10 * x10) + (y10 * y10)) ;
              cosinus := x10 / r ;
              invcos := 1 / cosinus ;
              sinus := sqrt(1 - cosinus) ;
              r10 := 0 ;
              while ((x20 <> x2) and (y20 <> y2)) do
                begin
                  r10 := r10 + invcos ;
                  x20 := round(r10 * cosinus) ;
                  y20 := round(r10 * sinus) ;
                  pencolor(white) ;
                  avance(x20,y20,1) ;
                  pencolor(none) ;
                end ;
            end ;
          moveto(x1 + 5,y1 + 5) ;
          pencolor(white) ;
          moveto(x2 + 5,y2 + 5) ;
          pencolor(none) ;
        end ;

    begin
      rep := 'Y' ;
      while (rep <> 'N') do
        begin
          init ;
          readln(rep) ;
          mtext ;
          write('QUELLE LIGNE DE DEPART -->  ');
          readln(y1) ;
          write('QUELLE COLONNE DE DEPART -->  ');
          readln(x1) ;
          write('QUELLE LIGNE D ARRIVEE -->  ') ;
          readln(y2) ;
```

```
        write('QUELLE COLONNE D ARRIVEE -->  ') ;
        readln(x2) ;
        graf ;
        aller(x1,y1,x2,y2) ;
        readln(rep) ;
        mtext ;
        write ('VOULEZ-VOUS RECOMMENCER ?  ') ;
        readln(rep) ;
        page(output) ;
    end;
end.
```

```
        .macro pop
        pla
        sta %1
        pla
        sta %1+1
        .endm


        .proc avance,3

return  .equ 0

        pop return
        pla
        tax
        pla
        pla
        sta 0c
        pla
        pla
        sta 0a
        pla
        sta 0b
        lda #255.
        sta 8
        lda #25.
        sta 9
        lda #0
        sta 3b

        lda 0c
        and #7
        sta 11
        lda 0c
        lsr a
        lsr a
        lsr a
        sta 10
        lda #7
        sec
        sbc 11
        sta 12
        lda 0c
        cmp #64.
        bcc un
        cmp #128.
        bcc deux
        lda #17
        sec
        sbc 10
        sta 13
        jmp xcalcul
un      nop
        lda #7
        sec
        sbc 10
        sta 13
        jmp xcalcul
deux    nop
        lda #0f
        sec
        sbc 10
```

```
        sta 13

xcalcul nop
        ldy 13
        lda #0
xxcal   cpy #0
        beq xxfin
        clc
        clv
        adc #80
        bvs xxovers
xxoverc dey
        imp xxcal
xxovers inc 3b
        imp xxoverc
xxfin   sta 3a
        ldy 12
        lda #0.
xycal   cpy #0.
        beq xyfin
        clc
        adc #4
        dey
        imp xycal
xyfin   sta 3d
        lda 0c
        cmp #64.
        bcc adun
        cmp #128.
        bcc addeux
        lda 3b
        clc
        adc 3d
        clc
        adc #20
        sta 37
        lda 3a
        sta 36
        imp ycalcul
adun    nop
        lda 3b
        clc
        adc 3d
        clc
        adc #20
        sta 37
        lda 3a
        clc
        adc #50
        sta 36
        imp ycalcul
addeux  nop
        lda 3b
        clc
        adc 3d
        clc
        adc #20
        sta 37
        lda 3a
        clc
        adc #28
        sta 36
```

```
ycalcul nop
        ldy #0
        lda 0b
        cmp #0
        beq yloop
        lda 0a
        clc
        adc #5.
        sta 0a
        ldy #36.
yloop   lda 0a
        cmp #7
        bcc yfin
        sec
        sbc #7
        sta 0a
        iny
        jmp yloop
yfin    sta 20
        lda #7
        sec
        sbc 20
        sta 21
        cpx 21
        bcc demiby
byteone lda 21
        cmp #0
        beq fbyteun
        lda (36),y
        sec
        rol a
        sta (36),y
        dec 21
        dex
        jmp byteone
fbyteun lda 20
        cmp #0
        beq suite
        clc
        lda (36),y
        rol a
        sta (36),y
        dec 20
        jmp fbyteun
demiby  lda #7
        sec
        sbc 20
        sta 24
        lda #7
        sec
        sbc 24
        sta 22
ddemiby cpx #0
        beq fdemiby
        lda (36),y
        sec
        rol a
        sta (36),y
        dex
        jmp ddemiby
fdemiby lda 22
```

```
          cmp #0
          beq done
          lda (36),y
          clc
          rol a
          sta (36),y
          dec 22
          jmp fdemiby

suite     iny
          cpy #40.
          beq done
          cpx #7
          bcc byplus
          lda (36),y
          ora #7f
          sta (36),y
          stx 24
          lda 24
          sec
          sbc #7
          sta 24
          ldx 24
          jmp suite
byplus    cpx #0
          beq done
          lda (36),y
          sec
          rol a
          sta (36),y
          dex
          jmp byplus

done      nop
          lda return+1
          pha
          lda return
          pha
          rts


          .proc init

          .public x1,x2,color

return    .equ 0
          pop return
          lda #0
          sta 40
          lda #20
          sta 41
          ldy #0

debinit   lda (40),y
          and #0
          sta (40),y
          lda 40
          cmp #0ff
          beq over
          inc 40
          jmp debinit
over      lda #0
```

```
        sta 40
        inc 41
        lda 41
        cmp #40
        beq fininit
        jmp debinit
fininit nop
        ldy #0          ; allows graphics
        lda #54
        sta 42
        lda #0c0
        sta 43
        lda (42),y
        lda #57
        sta 42
        lda (42),y
        lda #52
        sta 42
        lda (42),y
        lda #50
        sta 42
        lda (42),y

        lda #140.
        sta x1
        lda #96.
        sta x2

        lda #0
        sta color

        lda return+1
        pha
        lda return
        pha
        rts


        .proc mtext

return  .equ 0
        pop return
        ldy #0
        lda #54
        sta 42
        lda #0c0
        sta 43
        lda (42),y
        lda #51
        sta 42
        lda (42),y

        lda return+1
        pha
        lda return
        pha
        rts


        .proc graf

return  .equ 0
```

```
pop return
ldy #0
lda #54
sta 42
lda #0c0
sta 43
lda (42),y
lda #57
sta 42
lda (42),y
lda #52
sta 42
lda (42),y
lda #50
sta 42
lda (42),y

lda return+1
pha
lda return
pha
rts

.end
```