MASTER EN SCIENCES INFORMATIQUES

Didacticiel d'aide à l'évaluation d'une expression numérique

Nicolas, Ph.

Award date:
1994

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix

Namur

Institut d'informatique

Didacticiel d'aide à l'évaluation d'une expression numérique

Mémoire présenté par Ph. Nicolas en vue de l'obtention du titre de Licencié et Maître en informatique.

Promoteur : Cl. Cherton.

Année académique 1993-1994

Remerciements

Je tiens à remercier Monsieur C. Cherton d'avoir accepté d'être le promoteur de ce mémoire. Sa grande disponibilité, sa patience et ses conseils m'ont permis de mener à bien la conception de ce didacticiel.

Plan général

1 Introduction

Cette partie situe le problème dans le contexte général de l'apprentissage

2 Le projet

On trouve ici le détail des objectifs retenus pour construire le logiciel

3 Analyse fonctionnelle

Les grandes fonctions du logiciel sont décrites et analysées.

4 Analyse organique

La modularisation est détaillée.

5 Implémentation

Il y est expliqué comment on a réalisé les fonctionnalités.

6 Guide de l'élève

On y trouve les quelques indications nécessaires pour manipuler le logiciel.

7 Conclusions

Cette partie contient comme conclusions les extensions possibles des fonctionnalités du didacticiel.

8 Index

Il s'agit de l'index de termes employés dans un sens précis.

9 Table des prédicats

Liste exhaustive des prédicats triés par ordre alphabétiques avec référence de la page où ils se situent.

10 Bibliographie

11 Le code

Table des matières

Première partie

Le sujet et son contexte

---

1. Objet

Nous allons présenter le sujet et le contexte général dans lequel il se place. Ce contexte sera analysé, tant au point de vue de l'enseignement que de l'environnement informatique. Il sera ainsi possible de situer notre travail puis de cerner les caractéristiques que nous voulons lui donner.

2. Le sujet

2.1. Le sujet proprement dit

Il s'agit de mettre au point un didacticiel qui a pour objectif d'aider des élèves de l'enseignement secondaire à maîtriser l'évaluation d'une expression numérique, aussi complexe soit-elle, ne faisant intervenir que les quatre opérations.

Aucune directive particulière n'est donnée tant sur la méthode que sur le langage.

2.2. Cerner le sujet

Cette tâche est moins banale qu'il n'y paraît à première vue. En effet, on découvrira que bien des choix et des difficultés sont sous-jacents à ce type de problème.

Celui-ci, tel qu'il est présenté, est très ouvert. On peut même qualifier sa définition de "floue" avec les avantages et inconvénients qui y sont liés. Si on jouit d'une grande liberté de choix, il faut aussi les faire de manière judicieuse et les assumer. La première chose serait donc de préciser la tâche exacte du logiciel. Soulignons qu'il s'agit d'un travail créatif. La maturation des idées est lente. Certains ne voient, dans cette phase, qu'un préliminaire à écourter. C'est en réalité un moment décisif. Je ferai largement usage de mon expérience personnelle d'enseignant en mathématiques pour arrêter la stratégie utilisée dans ce logiciel.

3. Les acteurs

Il y a trois acteurs : l'élève, le professeur et l'informaticien. Il faut attirer l'attention sur le fait qu'ici, le rôle du professeur et de l'informaticien sont tenus par une seule et même personne. Je suis donc mon propre client.

4. Le problème

4.1. Les difficultés des élèves

4.1.1. Faible maîtrise de l'algèbre fondamentale

4.1.1.1. Le constat

On observe dans l'enseignement secondaire beaucoup de faiblesses dans la maîtrise du calcul d'expressions complexes et cela, parfois jusqu'en dernière année pour certains élèves. Ceux-ci se voient donc bloqués par de purs problèmes de calcul alors qu'ils sont en train de faire l'étude d'une fonction ou d'évaluer une intégrale. C'est surtout frappant lorsqu'il faut manipuler des fractions ou lorsque plusieurs niveaux de parenthèses interviennent.

Par exemple, l'évaluation de l'expression

$$2/3+5-(5/9)*(4+2/(9-1/3))-(4+5)*(4/5)$$

réserve parfois bien des surprises. Le mauvais usage des parenthèses, des confusions entre les règles, ... peuvent parfois expliquer ces égarements.

Plus simplement, dans les premières années, on voit certains élèves affirmer que

$$5+3*2=8*2=16$$

4.1.1.2. Les causes

Mon expérience personnelle me conduit à voir les origines suivantes aux problèmes rencontrés[1],

- Une mauvaise connaissance des règles de calcul. C'est évidemment la condition sine qua non pour maîtriser les opérations nécessaires à l'évaluation d'une expression. Il s'agit ici, pour l'élève, de fournir un effort systématique de mémorisation dont il ne voit pas toujours l'utilité.
  Par exemple, il faut savoir qu'un signe moins précédent une fraction dont le numérateur est une somme peut se distribuer au numérateur de cette fraction.
  Ainsi on a $-\frac{5+6}{9}=\frac{-5-6}{9}$.

- Pas de référence systématique à l'emploi des règles. Trop souvent, l'élève en difficulté travaille à l'intuition, au flair, comme il le sent ... Autrement dit, il n'établit pas de lien entre un calcul qu'il réalise et les structures des expressions. Un excellent exercice pour illustrer ce point de vue est d'obliger l'élève à justifier chaque ligne écrite. Souvent il est incapable de justifier sa démarche. Ceci est vrai non seulement pour les élèves en difficulté mais aussi pour la majorité des autres !

- Un mauvais usage des parenthèses. J'ai souvent vu dans des classes de quatrième année 3*5+4 au lieu de 3*(5+4) car les deux expressions se lisent de la même manière à la ponctuation près.

---

[1] On ne considère pas ici les facteurs qui ne relèvent pas strictement du cours tels les influences sociologiques.

- La grosse difficulté chez certains, c'est qu'ils ne voient pas la structure sous-jacente aux expressions. Ils ne peuvent donc pas faire de lien ni, a fortiori, se référer aux formules. Combien de fois ne voit-on pas un élève qui hésite dans le choix de la formule à appliquer quand il faut dériver une expression parce qu'il faut d'abord voir si on a affaire à un quotient, une somme, ...

- De mauvaises analogies. Un enseignant n'est pas étonné de voir
  $2*(5*6)=(2*5)*(2*6)$
  puisque, dit l'élève,
  $2*(5+6)=2*5+2*6$.

- Généralisation abusive à partir du premier exemple rencontré.
  Si on a $(2/3)*(7/8)=14*24$ alors certains élèves généralisent au cas où le * est remplacé par un +.

- Une mauvaise organisation des notes de cours. Si la présentation laisse à désirer la "lisibilité" est parfois telle qu'elle engendre des erreurs par elle-même.

En fait, ces erreurs cachent souvent un mal plus profond : la perte du sens des opérations effectuées. En effet, l'élève fait les calculs automatiquement, réagit par réflexe, sans même avoir la possibilité de réfléchir puisqu'il ne perçoit pas la signification de ce qu'il fait. Cette **information manquante** est un drame car elle fait apparaître la matière sous un jour rébarbatif, sans fondement. En plus les élèves éprouvent des difficultés à "retenir" cette matière puisque tout est alors basé sur la mémorisation.

4.1.2. La norme du groupe

Les classes aujourd'hui sont souvent hétérogènes. Or le rythme, en classe, est le rythme du groupe. Le niveau de difficulté est aussi celui du groupe. Chaque fois, il s'agit d'un compromis. Donc l'élève ne trouve pas toujours ce qui est exactement à sa mesure. Cet aspect des choses va croissant dans l'évolution de la structure de l'enseignement actuel.

4.2. Les carences du système

Depuis de nombreuses années déjà la tendance est de favoriser l'enseignement individualisé. Cette idée se concrétisait par la constitution de petits groupes. En réalité ce n'est plus, dans de nombreuses écoles qu'un vœu pieux. Il faut bien constater que le professeur fait de l'enseignement de masse lorsqu'il a devant lui vingt ou vingt-cinq élèves. Il lui est en effet bien difficile d'individualiser réellement son enseignement. Envisager des exercices justes à la mesure de chaque élève est lourd à gérer voire impossible à réaliser avec les moyens dont il dispose. C'est a fortiori vrai si on envisage la possibilité pour l'élève de créer ses propres exercices. L'ambition du logiciel est d'aider le professeur dans cette tâche.

4.3. Les besoins du professeur

Il serait utile qu'il dispose, pour chaque élève, de renseignements précis sur ses faiblesses et ses points forts. De plus, un diagnostic régulier devrait pouvoir être fait. En d'autres termes, c'est un bon dossier détaillé par élève qu'il faudrait. Tâche bien fastidieuse et mangeuse de temps pour le professeur mais que l'on pourrait espérer pouvoir générer automatiquement.

Mettre à la disposition des élèves une banque d'exercices où il va puiser à son rythme ou qui lui sont proposés en fonction de l'évolution de son apprentissage, est tout à fait réalisable également.

4.4. Les pré-requis

Les pré-requis sont élémentaires : la connaissance des quatre opérations, l'usage des parenthèses, les ordres de priorités des opérations élémentaires, les règles d'associations, de distributivité et de commutativité. Bien que d'apparence simple, c'est l'imbrication de tous ces points additionnés à la perte de sens qui rend la manipulation des expressions difficile à maîtriser pour les 12-18 ans.

4.5. Les types d'évaluation

Apprendre suppose qu'à un moment ou à un autre une évaluation sera faite ne fût-ce que par soi même. Il y a diverses manières d'évaluer les élèves. Elles sont loin d'être neuves. Aujourd'hui elles sont formalisées davantage, mais les concepts sont connus depuis longtemps. On peut classer les évaluations comme suit :

4.5.1. Evaluation normative

Le principe est ici de placer l'évaluation de l'élève dans le cadre du groupe et d'y faire référence. Le principe de classer les élèves suivant les points décroissants et de comparer à une moyenne de classe en est un exemple. Il y a donc à la base une comparaison d'un élève par rapport aux autres élèves.

4.5.2. Evaluation certificative.

Le principe est ici d'évaluer un élève et de faire un constat sans plus. Ce constat n'est accompagné d'aucune explication et n'est pas exploité pour modifier la suite de l'apprentissage. Les examens de fin d'année sont de l'évaluation certificative.

4.5.3. Evaluation formative.

Citons un tout récent document du Ministère de l'Education, qui explique ce qu'est ce type d'évaluation :

> Le but est de communiquer de l'information utile, d'indiquer les moyens de progresser, d'identifier les lacunes à combler et non pas de comptabiliser pour établir des totaux et des moyennes [3][2].

On peut aussi consulter G. De Landsheere qui donne de l'évaluation formative une définition équivalente [4].

Nous allons voir dans le paragraphe suivant l'utilité de distinguer ces divers types d'évaluation.

5. L'existant

5.1. Typologie et exemples.

On peut trouver dans le commerce quelques didacticiels destinés aux études primaires et aux premières années du secondaire. Plus rarement pour des niveaux supérieurs.

On peut distinguer les catégories suivantes :

- Logiciels d'information pure :

---

[2] Les numéros entre crochets renvoient à la bibliographie.

Par exemple : MacGlobe, Worldstack.
On peut les voir comme un navigateur dans une banque de données. Ces didacticiels sont souvent bien faits. La qualité de l'interface est primordiale.

- Logiciel de simulation :

  Par exemple :
  - Interactive Physics.
  - Cabrigéomètre
  - The geometer's sketchpad
  - FlightSimulator
  - SimCity

  On fixe des paramètres et on constate les changements produits. Par exemple, Interactive Physics est un logiciel interactif d'expérimentation des phénomènes physiques élémentaires. C'est un petit laboratoire.

- Logiciel d'apprentissage de réflexes :

  Par exemple:
  - Raisonnons
  - Au pays des fractions
  - Math drill
  - Add

  L'objectif est de faire un bilan, une évaluation des connaissances à un moment donné. Par essence, il ne s'agit pas ici d'expliquer la source des difficultés. L'ordinateur ne réagit pas dans les exercices proposés aux résultats des exercices précédents. L'évaluation mise en jeu est l'évaluation certificative.

5.2. Les critiques

Actuellement, du point de vue didacticiel, les meilleurs, à mon avis, sont ceux qui font de la simulation. Ils permettent de visualiser l'effet d'un changement de paramètres de façon concrète. Ils aident l'élève à développer une intuition du domaine étudié et à faire le lien entre la réalité de ce domaine et le formalisme utilisé à sa description.

Examinons maintenant les logiciels qui traitent de l'évaluation d'une expression numérique. Ils ont tous pour caractéristique principale qu'ils constatent l'échec ou la réussite de l'opération proposée. L'idée principale semble être que le "drill" est le moteur exclusif de l'apprentissage. On y constate aussi peu de souplesse quant à la complexité des expressions possibles. Enfin, chose étonnante, je n'en ai pas vu qui mémorise, en quittant l'application, autre chose qu'un score! L'aspect ludique semble être l'attrait majeur et souvent, ils sont d'une pauvreté affligeante. Ils font donc de l'évaluation certificative car l'ordinateur ne tient pas compte explicitement du résultat des exercices précédents.

Notre ambition est autre. On aimerait réaliser un logiciel qui ferait de l'évaluation formative. Il faut donc tenir compte de l'historique de l'élève. C'est dans cette optique que voudrait s'inscrire le logiciel, c'est-à-dire dans une optique d'apprentissage où l'explication est présente et où l'attitude de l'élève face à un exercice influence la suite de cet apprentissage.

6. La bibliographie

6.1. Sur le problème

Aucune bibliographie sur le sujet précis qui nous occupe ne semble exister.

6.2. Ouvrages généraux

6.2.1. Sur l'apprentissage par l'IA

Quelques ouvrages parlent de l'application des techniques de l'I.A. à la recherche dans l'enseignement (e.a. [9], [10]). On y fait part d'expériences concernant divers sujets, depuis la factorisation d'une expression jusqu'à l'apprentissage du Pascal en passant par les exigences de l'interface. Je renvoie le lecteur intéressé à ces ouvrages.

6.2.2. Sur prolog

L'implémentation a été réalisée en Prolog. C'est pourquoi nous parlons de la bibliographie relative à l'usage de ce langage. Les plus importants restent "The Art of Prolog" et "The Craft of Prolog" de Shapiro [6], [7]; "Programming in Prolog" de Clocksin et Mellish [8], dans sa dernière édition, vaut la peine et peut même être un bon ouvrage pour débuter, car on y trouve les bases du langage.

7. Pourquoi l'ordinateur

7.1. L'Enseignement Assisté par Ordinateur

Commençons par distinguer d'une part l'enseignement programmé par ordinateur communément appelé EPAO (Enseignement Programmé Assisté par Ordinateur) ou parfois EAO et d'autre part l'Enseignement Intelligent Assisté par Ordinateur ou EIAO. L'enseignement programmé par ordinateur est avant tout un enseignement programmé. Cette notion est antérieure à celle de l'utilisation de l'ordinateur.

Pour illustrer la différence, voici un petit extrait de *L'enseignement programmé* de De Montmollin [5].

> L'enseignement programmé est une méthode pédagogique systématique, reposant sur des bases expérimentales. Cette méthode se caractérise par : la recherche d'un ordre de présentation efficace; l'adaptation au rythme de l'élève; la participation active de l'élève; la correction immédiate et point par point de l'acquis.

L'ordinateur est là comme support. Le lecteur intéressé trouvera dans ce livre une étude approfondie sur le sujet. On y découvrira ainsi que bien des méthodes et principes ne sont pas neufs. L'enseignement programmé était concrètement réalisé à l'aide de livres qui n'étaient pas lus séquentiellement ou de fiches. Certaines machines spécifiques ont même été inventées pour servir de support à de tels enseignements. Ce n'est que plus tard, suite au développement des ordinateurs, que ceux-ci ont été utilisés à cette fin.

Par contre l'EIAO recourt à l'ordinateur pour mettre en place des techniques qui ne sont pas envisageables sans lui.

On parle beaucoup d'enseignement assisté par ordinateur, mais on voit peu d'applications dignes de ce nom. Analyser les causes de cet état de chose n'est pas notre propos. Soulignons juste quelques points.

Des langages spécifiques dits "langages auteurs" ont même été développés dans le but de faciliter la création de cours programmés ayant l'ordinateur comme support. Les didacticiels réalisés à l'aide de ces langages sont extrêmement coûteux en temps de réalisation, plus contraignants qu'un manuel et, le plus souvent, d'une qualité pédagogique discutable.

Il y a une réelle difficulté intrinsèque à ce que l'ordinateur puisse "comprendre" la démarche de l'élève, détecter l'endroit où il se trompe, trouver la cause de l'erreur et non se contenter de constater une erreur dans le résultat.

Enfin le maigre retour commercial explique, du moins en partie, la faiblesse du développement dans ce secteur.

7.2. Apprentissage et gestion individualisée

L'ordinateur peut permettre cette gestion individualisée dont nous parlions plus haut. Il est en effet, capable de stocker, traiter et restituer de l'information. Les informations qu'il peut fournir sont nombreuses et variées.

Il permet aussi à l'élève de travailler à son rythme et en l'absence du professeur. Ce dernier peut alors se libérer et se concentrer sur une approche plus fondamentale et plus individuelle des élèves.

Mais il faut aussi que l'ordinateur fasse les "bons traitements".

7.3. L'ordinateur et le professeur

L'ordinateur comme outil pédagogique est peu présent dans les écoles. Le coût du matériel est parfois avancé pour expliquer cette relative absence, mais je crois qu'il s'agit d'une mauvais argument. Il serait préférable de dire, selon moi, que l'informatisation de la pédagogie n'est pas perçue comme prioritaire et n'a donc que peu de moyens.

Certains perçoivent l'outil informatique comme un danger, car ils y voient, à tort, un remplaçant du professeur. Un élément important, pour les jeunes adolescents, est la communication qui est présente dans la relation qu'ils ont avec le professeur. Il y a dans cette communication une dimension affective que ne peut donner un ordinateur. Par exemple, il est toujours remarquable de constater, dans la pratique, qu'une phrase, écrite dans un manuel, devient souvent plus compréhensible si elle est prononcée par un être en chair et en os. Ajoutons aussi qu'un enseignement qui serait basé essentiellement sur l'ordinateur n'aurait plus la dimension sociale qu'il a actuellement et qui est reconnue comme un aspect primordial.

Il faut donc concevoir l'outil informatique comme un complément et non comme un substitut du professeur. Notons que ce danger n'est pas propre à l'outil informatique. Déjà pour l'enseignement programmé les mêmes dangers étaient signalés en son temps. Consulter à ce sujet [5].

L'ordinateur n'est pas un concurrent, mais un moyen supplémentaire dans la pédagogie moderne. C'est comme tel qu'il doit être considéré. C'est tout à fait analogue à la place de la calculette dans les cours scientifiques. Elle est loin de tirer d'affaire l'élève en difficulté. C'est un outil très utile. Mais elle ne reste qu'un outil. C'est vrai aussi pour l'ordinateur.

8. En résumé

Beaucoup de logiciels qui traitent de l'évaluation des expressions numériques font de l'évaluation certificative, c'est-à-dire qu'il font un constat du savoir-faire de l'élève. Ils ne permettent pas de réaliser un apprentissage où l'explication joue un rôle moteur important.

L'organisation de l'enseignement secondaire ne permet pas, en pratique, de faire un enseignement individualisé.

Les élèves font peu de liens entre l'expression qu'ils ont devant eux et les règles qu'ils doivent appliquer. Plus inquiétant, quand les règles sont mémorisées, il arrive que le sens de ces règles reste obscur. Ajoutons encore qu'ils sont souvent incapables de dire quelles

sont leurs propres faiblesses. Pour évoluer, il faut aussi se connaître soi-même. Il faudra donc informer au mieux les élèves sur leurs points faibles et le sens des règles à utiliser.

Deuxième partie

Le projet

1. Objet

Nous allons tracer les grandes lignes de notre projet à l'aide des conclusions du chapitre précédent. Ce sont les grands principes qui seront la charpente du logiciel.

2. Sa philosophie

La philosophie du projet est en quelque sorte une tentative de définir le grand absent dans les logiciels déjà existant dans le commerce : une stratégie pédagogique.

2.1. Réaliser une analyse

Nous voulons construire un outil qui va au-delà d'un simple constat relatif au résultat final d'un exercice. car un tel constat ne suffit pas à faire progresser dans la compréhension. Il doit permettre de réaliser une analyse des difficultés des élèves et donc personnaliser les sessions pour chaque élève en fonction de ses réactions aux sollicitations de l'ordinateur. C'est une application de l'esprit de la pédagogie différenciée.

2.2. Favoriser un progrès

L'enseignement est un travail de longue haleine. Un progrès à moyen et long terme n'est réalisable que s'il y a mémorisation de certaines informations utiles. Ainsi, lors d'une nouvelle session d'exercices, on ne recommence pas à zéro. Quand on parle de mémorisation, on entend par là une mémorisation d'une certaine richesse qui ne se limite pas à un score ! Permettre un progrès suppose aussi que l'on envisagera non seulement les faiblesses mais aussi les points forts.

2.3. Rôle du professeur

Il doit rester le gestionnaire de l'apprentissage. Il faut donc lui permettre d'intervenir dans un certain nombre de choix. Ceci se fera via les paramétrages pour adapter le logiciel au cas particulier de sa classe. Citons, par exemple, la possibilité pour l'élève de choisir des exercices préparés par le professeur en fonction du niveau de la classe, mais aussi la possibilité pour le professeur d'imposer le mode de dialogue avec l'élève. (Voir plus loin).

3. Glossaire

Dans la suite du travail, un certains nombre de termes seront employés dans un sens très précis. Le lecteur trouvera un petit index à la fin du mémoire afin de situer facilement l'endroit où se trouve la définition ou l'explication recherchée. Pour attirer l'attention du lecteur et pour un premier contact, cette liste est déjà reprise ci-dessous avec une définition succincte.

- **Structure** :

  la structure d'une expression est cette expression dans laquelle on fait abstraction des valeurs numériques (Voir p 20)

- **Famille** :

  Pour cette notion, il est préférable de consulter sa définition page 23.

- **Noyau** :

  C'est une famille présente au moins trois fois. (Voir p 25).

- **Expression où naît l'erreur**

  C'est une expression évaluée incorrectement, mais dont les deux sous-expressions sont évaluées correctement (Voir p 17).

- **Forme**:

  C'est un groupe d'informations pertinentes qui seront stockées sur disque pour assurer le suivi d'une session d'exercices à l'autre (voir p22).

- **Connaissance**

  C'est un réel qui représente le degré de maîtrise que élève a d'une structure (voir p20).

- **Décomposition manuelle**

  Action de sélectionner ce que l'on veut évaluer comme sous-expression (voir p29)

- **Recherche descendante**

  Action d'analyser un exercice résolu sur papier à partir de l'énoncé vers la réponse finale (voir p27)

- **Recherche montante**

  Action d'analyser un exercice résolu sur papier à partir de la réponse vers l'énoncé (voir p27).

- **Fausse règle**

  Une fausse règle est une règle que certains élèves appliquent croyant qu'elle est correcte alors qu'elle ne l'est pas (voir p17).

Dans la suite, un terme sera en gras lorsqu'il est défini et en italique quand il est employé dans le sens donné dans la définition.

4. La stratégie

Comment faire progresser l'élève ?
Il n'existe aucune recette miracle. On peut néanmoins dégager quelques idées.

4.1. Analyse de l'erreur

Pour chaque exercice, il faut rechercher la cause et la source de l'erreur. Ce qui est recherché n'est pas un traitement symptomatique mais une recherche profonde des causes.

Par exemple, considérons l'expression ((2+5)*6)/(2/7-1/4). qui est mal évaluée par l'élève. L'analyse peut conduire au résultat suivant :

la valeur globale de l'expression est fausse.
l'expression (2/7-1/4) est bien évaluée
l'expression ((2+5)*6) est mal évaluée
l'expression 2+5 est bien évaluée

On en déduit que la source de l'erreur est dans l'expression ((2+5)*6). Nous dirons que (2+5)*6 est **l'expression "où naît l'erreur"**. Mais il est impossible de préciser davantage la cause de l'erreur si je ne sais pas comment l'élève a réalisé son calcul. En effet, il peut s'agir d'une erreur de distribution ou d'une erreur de multiplication. Par contre, une information supplémentaire peut parfois aider à déterminer cette cause. Si la réponse de l'élève est 32, il y a de grandes chances qu'il s'agisse d'une erreur de distribution assez répandue : (2+5)*6=2+5*6. Il est donc essentiel que l'expert donne à l'ordinateur la liste des **fausses règles** appliquées par certains élèves. Ceci est détaillé dans le paragraphe Confusions. Nous verrons plus loin comment mettre en œuvre cette politique.

4.2. Analyse globale des résultats

On entend par là, le fait de présenter une vue d'ensemble de l'activité d'un élève. Il faut à cet effet, mémoriser une série d'informations attachées à chaque exercice, et cela, pour chaque élève. Il faut donc définir les informations pertinentes. C'est en quelque sorte une évaluation permanente.

4.3. Aide au raisonnement

L'élève doit être amené à se raccrocher uniquement aux règles existantes. Il doit donc être capable d'abstraire. Pour les exercices où il se trouve en difficulté, on souhaite pouvoir lui montrer les formules mal connues. Nous avons déjà cité le cas d'une mauvaise distribution. Citons encore le cas suivant : si l'élève fait -(2/9-5*2)=-2/9-5*2, cet élève doit faire le lien avec -(a-b)=-a+b.

4.4. Les interactions lors d'une série d'exercices

Le projet va appliquer une conception d'un logiciel d'aide à l'apprentissage où les idées de bases peuvent se résumer comme suit :

- Le déroulement d'un exercice est analysé par l'ordinateur qui emmagasine les informations relatives à cet exercice. Cette information va agir sur les exercices suivants. Il y a donc interaction entre la création d'un exercice par l'ordinateur et les informations dont dispose l'ordinateur.
- Les commentaires vont influencer la réponse de l'élève aux exercices suivants et cette réponse va agir sur l'analyse que fera l'ordinateur.

Ces idées sont schématisées dans la figure 1.

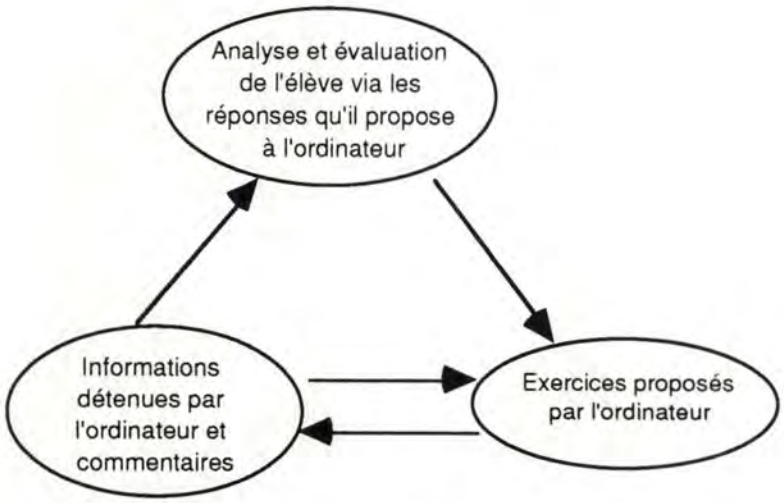


Figure 1

Cheminement des interactions au cours d'une serie d'exercices

5. Le déroulement du développement

5.1. La nature du sujet

Le sujet tient à la fois de la gestion et de l'intelligence artificielle En effet, pour la gestion, citons par exemple la mise à jour permanente des fichiers élèves. Nous verrons d'autres exemples plus loin. Pour l'intelligence artificielle, commençons par préciser ce que l'on entend par là. Pour Farrenhy et Ghallab [1]:

> Le concept d'intelligence artificielle est complexe et relatif, et l'expression, fortement controversée, d' "intelligence Artificielle" ne le précise pas davantage. Aussi est-il difficile de définir la discipline scientifique qu'est l'IA. Comme le fait remarquer un spécialiste du domaine (A. Newel, cité dans [BARR and FEIGENBAUM]) cette difficulté n'est pas propre à l'IA : une discipline scientifique **ne se définit pas** (au sens d'un projet dont la spécification précède la réalisation), elle **se constate**.

et plus loin encore

> Le champ d'investigation de l'IA est le raisonnement : **l'IA cherche à comprendre les mécanismes de compréhension.**

Or, par expérience, je puis affirmer qu'une aide efficace à l'apprentissage de l'évaluation d'expression est intimement liée à la compréhension des principes sous-jacents et non à la mémorisation de "recettes". L'IA rentre donc bien dans le cadre du travail de ce mémoire si on se réfère à Farrenhy et Ghallab.

Il y a divers points de vue sur ce qu'est l'intelligence artificielle. On peut les voir comme complémentaires. Ainsi on dira aussi que l'IA est l'approche d'une solution sans que l'on puisse prévoir exactement son déroulement et sans savoir non plus si on va parvenir à trouver une solution [2].On peut classer dans cette catégorie la recherche de la nature de l'erreur de l'élève.

5.2. Le cycle de développement

En pratique, pour notre problème, l'approche "waterfall" n'est pas indiquée. Le travail qui nous occupe, plus que d'autres, se prête mal à une approche entièrement et strictement top-down. Voyons pourquoi.

Ma pratique professionnelle m'a appris que le problème est complexe. Bien davantage qu'il n'y paraît. Il n'est guère possible de concevoir d'une façon Top down un problème qui doit répondre à tellement d'attentes. Citons la représentation de la connaissance, la modélisation de l'interaction didactique, et en particulier les formes efficientes de prise en compte de l'apprenant, la modélisation des processus interactifs et du dialogue.

Des bonnes idées viennent aussi durant la conception et lors des essais avec des élèves. Il me semblait bien imprudent de ne pas se réserver la possibilité de changements suite aux réactions des élèves.

Aucun objectif précis n'est défini au départ

Au vu de l'ampleur des facteurs à prendre en compte, Il est apparu qu'il était raisonnable de pouvoir réaliser facilement du maquettage. Le choix du langage a été dicté par la volonté de pouvoir réaliser ce maquettage.

Le cycle de vie, dans le cadre d'un prototypage peut se schématiser comme suit :

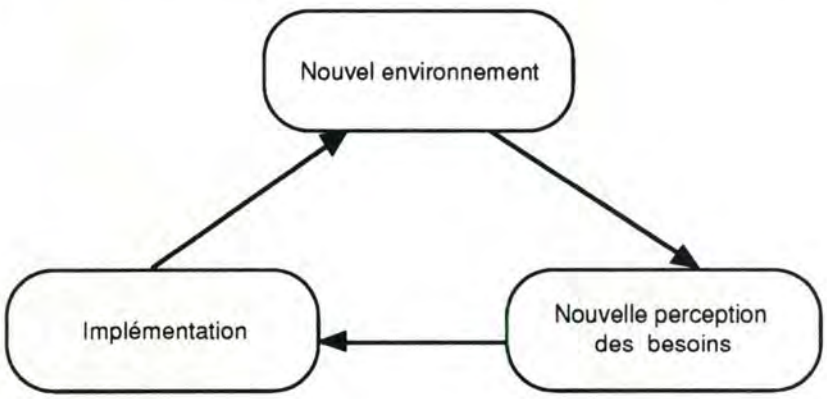


Figure 2

Cycle de développement

Il faut accepter l'idée qu'il n'y a pas de point vraiment final dans le développement d'un logiciel. Plusieurs boucles sont souvent nécessaires pour arriver à un développement "acceptable". Les boucles aident à penser la conception même du logiciel. Il faut considérer ce cycle comme un outil à penser.

5.3. Particularité du logiciel

Il faudra regarder aussi ce travail comme un laboratoire où les idées n'ont souvent pas été poussées jusqu'à leur exploitation complète. Quelques boucles seraient encore nécessaires. En fait, on a ouvert des portes sur des pistes intéressantes.

5.4. Évaluation du travail

Aucun objectif précis ne peut dans cette matière être facilement exprimé dans un cahier des charges. Et même si on le faisait, il n'y aurait pas nécessairement consensus sur

la pertinence de ces objectifs pour y voir un bon apprentissage de la matière[3]. On comprendra donc la difficulté à pouvoir mesurer si l'objectif que l'on s'était donné, à savoir aider les élèves dans leur apprentissage, est atteint et dans quelle mesure.

6. Les outils

6.1. La structure

Cette notion est essentielle et fondamentale dans ce travail. C'est un premier pas dans l'abstraction. La **structure** est l'architecture d'une expression numérique. En d'autres termes, c'est l'expression dépourvue de ses nombres. Nous symboliserons la place que peut occuper un nombre par un "_". Plusieurs expressions numériques différentes peuvent être des occurrences différentes de la même structure.

Par exemple, nous avons

Expression	Structure
4*(6-2)+5/9-1	_*(_-_)+_/_-_
6*(2-5)+8/3-5	_*(_-_)+_/_-_
(4-5/6)+6/(1+5)	(_-_/_)+_/(_+_)

Tableau 1

Les structures à partir des énoncés

6.2. La connaissance

6.2.1. Objectif poursuivi

Nous souhaitons mesurer, à l'aide d'un réel, le degré de maîtrise que possède un élève d'une structure donnée. Il faut donc pouvoir se référer à la suite des expériences passées liées à cette structure.

De façon tout à fait élémentaire, il suffirait de calculer, pour chaque structure, la moyenne des expériences passées . Mais cette façon de faire n'est pas indiquée. En effet, ce qui nous importe est

- de ne pas perdre la mémoire du passé;
- de faire primer le passé récent sur le passé lointain.

6.2.2. Définition.

La **connaissance** C d'une structure sera définie comme suit :

$$C=\mu Cr + (1-\mu).Ca$$

---

[3] Il n'est pas rare d'assister à des divergences de vue entre les membres de la communauté éducative sur les méthodes à suivre dans l'enseignement secondaire. Souvent, elles traduisent des divergences de conception sur l'objectif de l'enseignement et parfois, plus profondément et sans que ce soit exprimé, sur des valeurs de vie. En effet, pour certains, le but de l'enseignement est de "préparer à la vie", pour d'autre c'est de former des gens heureux. On oppose les deux très souvent. Pour les premiers, l'enseignement est au service de l'économique et a donc un caractère utilitaire. Pour les autres non.

où
Cr est la connaissance récente.
Ca est la connaissance ancienne
μ est le poids de la connaissance récente Cr dans le calcul de C

6.2.3. Calcul de la connaissance

Pour chaque structure rencontrée on mémorise, arbitrairement, la liste des dix derniers résultats (0=échec ou 1=réussite) ainsi que l'estimation de la *connaissance* C. Cette liste est donc un historique

Lorsque l'élève effectue une nouvelle évaluation E relative à une structure S, on ajoute en tête de liste le nouveau résultat, le dernier étant éliminé et

Ca est la connaissance relative à S avant l'évaluation E
Cr est la moyenne des dix derniers résultats relatifs à S compte tenu de E
C est la connaissance relative à S tenant compte de E.

6.2.4. Initialisation.

Au départ, lors de la naissance d'une structure nouvelle, il faut assigner une valeur initiale à

- C. Ce peut raisonnablement être 0, mais c'est discutable.
- μ. Son choix est affaire d'appréciation et d'expérience.
- La liste de départ constituée de dix dernières expériences bidons.
  Soit [0,0,0,0,0,0,0,0,0,0]. Ce choix est arbitraire et correspond au cas d'un élève qui débute dans son apprentissage.

6.2.5. Évolution.

Considérons le cas où, pour la même structure, toutes les réponses, depuis le début, sont correctes. Nous avons alors l'évolution suivante

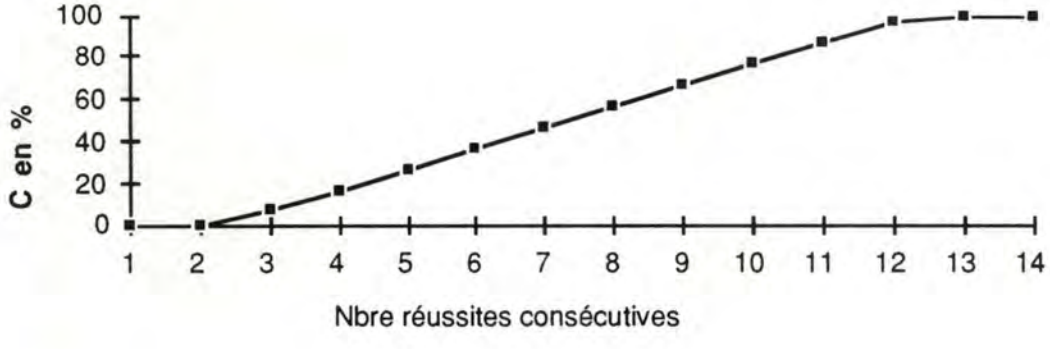


Figure 3

Evolution de la connaissance (mu=0,81)

On voit que la courbe tend asymptotiquement vers 100 %. Sa progression est quasi linéaire sur une portion du graphe.

6.2.6. Le choix de μ

En fait, si l'on augmente le poids de la connaissance récente Cr, l'évolution est plus rapide et colle mieux à l'état actuel de la maîtrise qu'en a l'élève. Mais elle est aussi plus sensible aux variations "locales".

6.3. La "forme"

Cet outil utilise les deux outils précédents.

A chaque *structure* on attache une série de renseignements :

- Son origine

  - soit l'élève rentre sa propre expression au clavier.
  - soit l'élève charge l'ordinateur de lui proposer une liste d'expressions déjà rencontrées par lui mais qu'il maîtrise mal. L'élève fait son choix dans cette liste.
  - soit l'élève choisit un exercice puisé dans un fichier préparé à cet effet par le professeur.

  Notons qu'un même exercice peut être rencontré avec deux origines différentes. Il peut être pertinent de conserver cette information. En effet, le professeur pourrait utiliser cette information pour vérifier si l'élève a utilisé le fichier qu'il a mis a sa disposition. Les origines peuvent être mémorisées sous la forme d'une liste de trois éléments Chacun de ces éléments représente le nombre de fois que l'expression a pour origine celle liée à cet élément .

- La liste des dix derniers résultats

  Voir supra.

- La valeur de la *connaissance*.

- Deux compteurs

  - un compteur indiquant le nombre de fois que l'évaluation d'une expression dont la *structure* est celle qui nous occupe fut correcte.
  - un compteur indiquant le nombre de fois que l'évaluation d'une expression dont la *structure* est celle qui nous occupe fut incorrecte.

  Ces deux compteurs seront utilisés pour estimer si une connaissance est pertinente. En effet, si, la connaissance d'une structure est faible et que le nombre de fois que cette structure a été rencontrée est petit (trois ou quatre fois par exemple) on ne peut en déduire que l'élève maîtrise mal la structure en question (voir plus loin le paragraphe relatif à l'établissement d'un rapport)

En résumé, la **forme** contient six renseignements

forme(Structure, Connaissance, Origine, Liste Résultats, CptBon, CptFaux).

6.4. Les fichiers individuels

Ce fichier contient toutes les *structures* des exercices réalisés par un élève donné quel que soit l'origine des exercices.

Sauver les *formes* relatives à un élève est essentiel puisqu'ainsi, les informations sont conservées et intégrées lors de la session suivante de cet élève.

Voici un exemple de fichier d'élève qui est un fichier texte :

```
forme(_186+_187+_183, 0.05, [1, 0, 1], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0], 1, 1) .
forme(_192*_193, 0, [1, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], 1, 0) .
forme(_202+_203-_199, 0.05, [1, 0, 1], [1, 1, 0, 0, 0, 0, 0, 0, 0, 0], 2, 1) .
forme(_208-_209, 0.265, [1, 1, 0], [1, 1, 1, 1, 0, 0, 0, 0, 0, 0], 4, 0) .
forme(_214*(_218-_219), 0.09600000000000000001, [1, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0], 1, 3) .
forme(_228/_229+_225, 0, [1, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], 1, 0) .
forme(_234*(_242/_243+_239), 0, [1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0, 1) .
forme(_248-_252/_253, 0.3625, [1, 1, 0], [1, 1, 1, 1, 1, 0, 0, 0, 0, 0], 5, 0) .
forme(_258*(_262-_266/_267), 0, [1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0, 1) .
forme(_276/_277-_273, 0, [1, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], 1, 0) .
forme(_282*(_290/_291-_287), 0, [1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0, 1) .
forme(_296+_297, 0.6603125, [1, 0, 1], [1, 1, 1, 1, 1, 1, 1, 1, 0, 0], 8, 0) .
forme(_302*(_306+_307), 0, [1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0, 1) .
forme(_316/_317+_320/_321, 0.08, [1, 0, 0], [1, 1, 0, 0, 0, 0, 0, 0, 0, 0], 2, 0) .
forme((_330-_331)*(_338/_339+_335), 0, [1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0, 2) .
forme(_344+_348/_349, 0, [1, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], 1, 0) .
forme(_354*(_358+_362/_363), 0, [1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0, 1) .
```

6.5. Les familles

La bonne expérience est celle qui nous fait connaître autre chose que le fait isolé; c'est celle qui nous permet de prévoir, c'est-à-dire de généraliser. C'est pourquoi, conduire l'élève vers l'abstraction est capital.

Je voudrais ajouter ici un nouvel élément. Donner un outil qui aide les élèves à atteindre cette abstraction rentre bien dans le point de vue actuel de lutte contre l'inégalité sociale parmi les élèves. Voici un extrait de la conférence de Ph. Meirieu donnée à l'U.L.B. le 9 octobre 1992 :

> Je pense aux travaux du sociologue Bernstein qui insiste sur le poids du langage, des codes qui, par l'intermédiaire du langage, déterminent la réussite ou l'échec scolaire. Il y a le code restreint, explique Bernstein, qui est le code utilisé dans les milieux populaires et qui est essentiellement à caractère utilitariste et le code élaboré qui permet d'accéder à l'abstraction et qui est utilisé très tôt avec des enfants de milieux plus favorisés.

Nous avons déjà réalisé un premier pas vers l'abstraction en dégageant la notion de *structure*.

6.5.1. La représentation conceptuelle d'une structure

Pour aborder la suite, il est nécessaire d'aborder la manière dont on va concevoir la représentation mentale d'une expression.

Une expression numérique sera pensée comme un arbre binaire. Par exemple, l'expression 4+5*(7-2) se représentée par

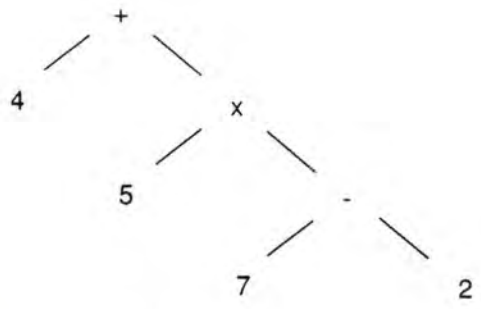


Figure 4

L'expression 4+5*(7-2) vue comme un arbre binaire

Lorsque nous avons affaire à un "-" monadique, on peut le représenter comme "0-"; Par exemple, -6 devient 0-6. On peut aussi considérer que l'arbre peut ne pas être binaire.

6.5.2. Créer une famille

Après un certain temps, la liste des expressions proposées sur base de la *connaissance* de l'élève peut devenir fort longue et, en théorie, illimitée. Il est intéressant d'essayer de dégager une synthèse en regroupant ces *structures* en "structures de base" que j'appellerai **famille**. Les critères de regroupement sont à définir. J'ai décidé, pour chaque *structure*, de ne conserver dans l'arbre qui la représente que les deux premiers niveaux en plus de la racine pour obtenir ainsi une structure de base. Il est clair alors que des *structures* différentes peuvent avoir la même structure de base. Nous aurons donc au maximum 101 configurations différentes[4], ce qui n'est déjà pas mal. Si on regroupe les opérateurs par type additifs et multiplicatifs, on n'aura plus que dix neuf structures de base. Dans ce cas on peut envisager de conserver trois niveaux pour chaque arbre. Mais ceci n'est pas indiqué en pratique car le signe moins implique des règles différentes du signe plus (par exemple dans la distributivité). C'est aussi le cas pour / et * pour la commutativité.

Voici quelques exemples de structures de base dégagées à partir des *structures*.

1+(5-1/2)	a+(b-c)
(4+5)*7	(a+b)*c
72+(2/3-4)	a+(b-c)
5-(3-5/6+2+1)	a-(b+c)
(2+1/3)*5(7/2)*(8+1)	(a*b)-(c*d)

Tableau 2

Exemple de familles dégagées à partir d'énoncés

---

[4] On a les 5 situations suivantes avec quatre opérateurs:
(a op_1 b) op_3 (c op_2 d) : 4^3 = 64 configurations
a op_3 (c op_2 d) 4^2 = 16 configurations
(a op_1 b) op_3 c 4^2 = 16 configurations
a op_3 b 4 configurations
a 1 configuration
donc au total 101 configurations

6.5.3. Utilité

Les *familles* font apparaître les *structures* à risque. Il est intéressant de retenir les structures de base correspondant aux structures où "naît l'erreur" dans la résolution. On a donc une meilleure connaissance des faiblesses, car on détecte ce qui lie un certain nombre d'entre elles.

6.5.4. Application

On propose aux élèves de pouvoir afficher ces structures dans une liste à l'écran. L'élève en choisissant de faire un exercice de cette liste va au coeur de ses difficultés.

Sur un exemple nous avons :

Les étapes	Transformations successives
Expression	2+5*(7-1/2)
Structure	_+_*(_-_/_)
Famille	a+b*c

Tableau 3

Les transformations successives

6.6. Le noyau

Lorsqu'une expression est rencontrée pour la première fois, elle peut donner naissance à elle toute seule à une *famille.* L'existence d'une *famille* n'est pas le signe d'une faiblesse. Mais la répétition de la présence de structures ayant une connaissance faible et appartenant à une même *famille* peut être interprétée comme le signe d'une faiblesse commune sous-jacente. J'appellerai **noyau**une *famille* issue d'au moins trois structures différentes. Ce nombre est arbitraire, mais c'est une première approche. On peut alors le voir comme le coeur des difficultés de l'élève. Il suffit donc à partir des *structures* dont la connaissance est faible de construire la liste de ces structures dont on n'a conservé que les deux premiers niveaux en plus de la racine et ensuite de retenir les éléments au moins en triple.

A titre d'exemple, nous pourrions avoir

Enoncés

3*(5-2)
4*(9/2-4)
3/2+6
8*(3-1/(5+2))
2*9-2
5/6+9/7

Les familles correspondantes à cette liste sont

a*(b-c)
a/b+c
a*b-c

a/b+c/d

Le noyau est alors

a*(b-c)

En résumé, nous avons donc la progression suivante dont un *noyau* est l'aboutissement:

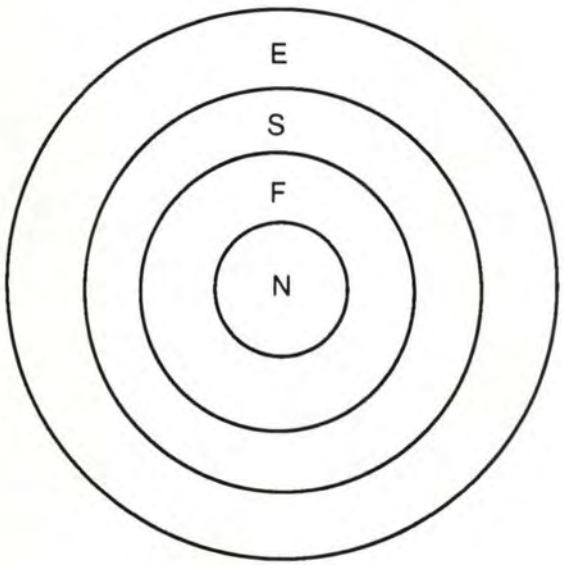


Figure 5

Cheminement vers l'abstraction

où N est le noyau, F est les familles, S est les structures et E les énoncés.

6.7. Les confusions

Il me semble impératif d'intégrer dans ce logiciel le fruit de l'expérience de l'enseignement. Je parle de la connaissance des faiblesses habituelles ou, si l'on veut, des erreurs généralement commises par les élèves. C'est une façon d'agir préventivement ou de pouvoir parfois mettre le doigt sur les maillons faibles du raisonnement connus des professeurs de mathématiques. Les principales confusions sont :

L'expression donnée	L'expression confondue
a*(b+c)	a*b+c
a*(b-c)	a*b-c
(a+b)*c	a+b*c
(a-b)*c	a-b*c
a-(b+c)	a-b+c
a-(b-c)	a-b-c
(a/b)/c	(a*c)/b
a/b+c/d	(a+c)/(b+d)

Tableau 4

Les principales confusions

J'exploiterai cette liste pour cerner au mieux la source des erreurs.

6.8. Consultation d'un aide mémoire

Il est utile de mettre à la disposition de l'élève un formulaire qui contient la liste des formules fondamentales de l'algèbre. Il faut qu'il puisse les consulter à tout moment.

6.9. Consultation d'une vue d'ensemble de ses résultats

J'envisage ici un rapport général qui donne un jugement sur la connaissance et son évolution pour chacune des *structures* que l'élève a eu l'occasion de manipuler et qui a fait l'objet d'une mémorisation par l'ordinateur.

7. Les moyens

7.1. Les types d'exercices

Je distingue cinq **types d'exercices** ou catégories d'exercices. On peut les classer comme suit :

7.1.1. L'élève prend l'initiative

7.1.1.1. Type 1

L'élève choisit de créer lui-même son exercice.
Un filtre peut signaler si cet exercice a une structure déjà bien connue de lui. L'élève a donc toute liberté pour la complexité de l'expression qu'il introduit dans le logiciel.
Ce type est qualifié d'exercices **personnels**

7.1.1.2. Type 2

L'élève demande à l'ordinateur de lui présenter une liste d'exercices
Cette liste n'est pas n'importe quelle liste. Elle est construite à partir des réponses de l'élève aux exercices précédents.

Les exercices seront choisis en sélectionnant ceux qui correspondent à une connaissance jugée faible. Ils sont donc de deux sortes :

- ceux dont la structure a déjà été rencontrée souvent et dont la connaissance liée à cette structure est faible.
- ceux tels que la structure n'a pas encore été souvent rencontrée et dont la connaissance de cette structure est donc faible par défaut.

Comme élément supplémentaire pour guider le choix de l'élève, chaque exercice de la liste, un par structure faible, est précédé de la valeur de la connaissance attachée à cette structure. La liste est triée sur la valeur de la connaissance. L'élève choisit ensuite un exercice dans cette liste en sachant le degré de maîtrise qui y est attaché. Ces exercices sont qualifiés d'exercices **"à travailler"**.

7.1.1.3. Type 3

L'élève demande à l'ordinateur de lui présenter la liste des exercices préparés par le professeur.

De nouveau ici, l'élève choisit un exercice de la liste. Ce type d'exercices est qualifié d'exercices **du professeur**.

7.1.1.4. Type 4

L'élève demande à l'ordinateur de lui présenter la liste des familles.

De nouveau ici, l'élève choisit un exercice de la liste. Ce type d'exercices est qualifié d'exercices **"mes bases"**.

7.1.2. L'élève est contraint ou sollicité.

7.1.2.1. Type 5

Dans ce cas, c'est l'ordinateur qui propose à l'élève un exercice sans avoir été sollicité par cet élève. Cette possibilité est signalée sans plus. Elle n'a pas été implémentée dans le logiciel.

7.2. Les modes de correction

Comment un enseignant corrige-t-il généralement un exercice du type de ceux qui nous occupent ? Deux possibilités s'offrent à lui.

La première manière consiste à lire le déroulement du développement. Il s'agit donc du parcours chronologique des différentes transformations effectuées par l'élève.

Une seconde manière consiste à commencer par la fin en vérifiant le résultat. On "remonte" en parcourant les transformations dans l'ordre inverse de l'ordre chronologique.

Dans chaque cas, ce qui est observé, c'est, en parallèle, l'évolution de chaque partie de l'expression qui a fait l'objet de transformations.

Si, dans les deux cas, le parcours est complet, l'analyse finale est la même. Chaque erreur peut être localisée précisément. De plus, le professeur est capable de voir les répercussions sur la suite des calculs et de ne pas considérer comme nouvelles erreurs celles qui ne sont que la répercussion logique de la première. Il fait un tri.

Par contre, si le parcours n'est pas complet, les deux parcours ne conduisent pas au même relevé des erreurs. Le parcours n'est pas toujours complet car il arrive qu'une erreur engendre des expressions qui sortent du cadre de l'exercice, qui ne rentrent plus dans le cadre du cours, ou qui n'auraient pas beaucoup de sens.

Nous appellerons **recherche descendante** la démarche qui consiste à partir de l'énoncé et **recherche montante** celle qui consiste à partir du résultat. Enfin, il y a un dernier mode de correction qui demande de la part du professeur une excellente connaissance

des faiblesses de l'élève et qui consiste à pointer directement les sources d'erreurs sans devoir tout examiner en détails. Cette démarche est la **recherche directe**.

Il est important aussi de ne pas perdre de vue que le professeur, quand il corrige, à devant lui la démarche complète de l'élève qu'il peut consulter comme il l'entend.

7.3. Exemples d'exercices

Comment un élève réalise-t-il un exercice ?

Rappelons que l'objectif de l'exercice est d'évaluer correctement une expression numérique. L'élève procédera par réduction successive de l'expression de départ. Examinons la manière dont peut s'effectue cette réduction.

L'élève peut remplacer une expression par une nouvelle expression jugée plus simple. Par exemple, une mise en évidence relève de cette politique. Les regroupements concernent des sous expressions qui ne sont pas nécessairement consécutives. L'élève peut aussi remplacer une expression directement par sa valeur et faire de la sorte un mini exercice à l'intérieur de l'exercice proposé. C'est souvent le cas lorsqu'il peut mentalement calculer une expression qui n'est pas trop compliquée. Voyons un exemple.

Soit les transformations successives suivantes :

```
2*[5+7*(4+7)]-(7-5)*5
2*[5+7*(4+7)]-2*5
2*[5+7*(4+7)-5]
2*[7*(4+7)]
2*7*11
14*11
154
```

Tableau 5

Un exercice

7.4. Connaître de bonnes informations

Nous avons signalé plus haut que constater que l'évaluation est incorrecte ne suffit pas toujours à déterminer le type d'erreur. Parfois, le simple examen de la valeur proposée suffit pour poser un diagnostic probable. (voir analyse de l'erreur). L'exemple suivant est plus complexe.

Soit l'énoncé 2*(5-2*3).
On peut observer les démarches suivantes proposées par l'élève,et cette liste n'est pas limitative :

a) 2*(5-6)
   10*11
   121

b) 2*(3*3)
   18

c) 2*5-2*3
   10-6
   4

Figure 6

Diverses démarches chez un élève.

Ici on voit qu'il serait judicieux de connaître la démarche de l'élève dans le détail pour différencier les types d'erreurs. Je suggère le diagnostic suivant :

a) erreur de distraction
b) 5-2*3 est devenu (5-2)*3. L'élève n'a pas respecté la priorité des opérateurs
c) Il n'y a pas eu de distribution.

Il est donc souhaitable de saisir au mieux la démarche complète de l'élève.

7.5. Les implications d'un mode de correction

Examinons de plus près ce qui se passe suivant le mode de démarche adoptée pour examiner l'exercice lorsque l'on s'arrête à la première erreur rencontrée. Considérons l'exercice de la figure 7.

Soit l'énoncé 2*(5-2*3)+5-(2+3). On peut avoir les transformations successives suivantes :

2*(5-2*3)+5-(2+3)	(1)
2*(5-6)+5-2+3	(2)
10-6+5-2+3	(3)
4+3+3	(4)
10	(5)

Figure 7

Un exercice

Si la recherche est descendante, on relève une erreur en passant de la ligne (1) à la ligne (2). Avec l'autre démarche on note une erreur entre les lignes (2) et (3). Ces erreurs ne sont pas de même nature.

Une recherche stoppée après avoir constaté la première erreur limite donc le nombre d'erreurs trouvées, mais de plus, cela peut conduire à ne pas observer les mêmes erreurs suivant la démarche adoptée.

7.6. S'inspirer du professeur

Dans notre projet nous essayerons de réaliser, dans la mesure du possible, les trois démarches observées.

Une première approche consistera à concevoir une expression comme un arbre dont la racine est l'énoncé. et d'y appliquer la *recherche montante.* On part donc de la réponse et, si elle est fausse, le sous arbre gauche sera évalué. Deux cas se présentent :

1 S'il est bon, on examine le sous arbre droit. Si ce dernier est bon à son tour, l'expression de départ est celle *où naît l'erreur.* Si ce sous arbre droit est faux, on le parcourt.
2 S'il est faux, on le parcourt.

A chaque étape, l'élève est questionné pour connaître la valeur qu'il donne à chaque sous expression qui lui est proposée. Nous parlerons aussi de recherche interactive.

Une seconde approche consistera à essayer de localiser directement l'erreur dans un sous arbre de l'expression à l'aide des renseignements dont dispose l'ordinateur. Nous parlerons de **recherche directe** ou aussi de recherche automatique.

Dans ces deux premiers cas, nous dirons que nous avons affaire à une **décomposition par l'ordinateur**.

La dernière approche, qui est pour moi la plus intéressante, consiste à laisser l'élève réduire l'expression comme il le souhaite. Nous parlerons de **décomposition manuelle**.

7.7. Critiques

7.7.1. Recherche montante

Cette fonction n'intervient que dans le cas où on laisse à l'ordinateur le soin de décomposer l'expression. Il est évident que c'est restrictif puisque l'élève peut avoir réalisé son calcul autrement. Mais, malgré son caractère arbitraire, cette démarche permet de détecter des faiblesses de l'élève. Il est important de noter que les faiblesses ainsi trouvées sont bien réelles et la démarche est donc valide. Sur une série d'exercices, on peut supposer que l'ensemble des faiblesses ainsi répertoriées sont bien représentatives de l'élève. Plus fondamentalement, la décomposition manuelle essaye de lever ces restrictions par une approche différente.

Une faiblesse de cette recherche est qu'elle peut avoir un caractère fastidieux si l'expression à évaluer est longue et si une seule erreur a été commise, celle-ci ayant été commise très tôt lors des transformations.

7.7.2. Recherche directe

L'intérêt de cette possibilité est d'épargner l'aspect fastidieux de la recherche d'une expression *où naît l'erreur.* La difficulté est qu'il faut savoir cibler avec une chance de succès très élevée l'endroit *où l'erreur naît*, sans quoi on en arrive à faire plusieurs suggestions qui, au pire, peuvent être toutes sans succès. Il faut alors revenir à notre bon vieux parcours et finalement la démarche complète risque d'être encore plus longue. Il n'est pas évident de cibler une *expression où naît l'erreur* avec de très grandes chances de succès, aussi cette démarche est à considérer avec prudence.

7.7.3. Décomposition manuelle

Nous réduirons le problème en le limitant de deux manières.

- une expression que l'on envisage de réduire ne sera remplacée que par une valeur (un entier ou une fraction).
- une expression ne sera effectivement remplacée que par sa VRAIE valeur.

Si l'élève propose une valeur fausse, le dialogue attendra que l'on propose la valeur correcte ou bien que l'on "donne sa langue au chat". La réponse finale est dans ce cas toujours la bonne. Il est alors clair que toutes les erreurs commises peuvent être mémorisées. D'autres démarches sont évidemment possibles.

Nous verrons plus loin qu'une importante restriction à cette méthode est l'impossibilité de réaliser des sélections multiples avec le logiciel employé.

7.8. Le paramètrage

Nous distinguons deux types de paramètrages : ceux qui sont accessibles à l'élève et ceux qui ne sont accessibles qu'au professeur.

Ceux qui sont accessibles aux élèves et qui déterminent

- le type de *décomposition* de l'expression pour analyser l'erreur. L'élève a en effet le choix de laisser l'ordinateur décomposer lui-même l'expression donnée ou de le faire lui-même.

- le type de *recherche* qui n'a de sens que si la décomposition est faite par l'ordinateur. Il est en effet possible de rechercher l'endroit ou se situe l'erreur de deux façons : la première est de décomposer systématiquement l'expression, l'autre est de laisser l'ordinateur effectuer une *recherche directe*.

Ceux qui sont des paramètres techniques uniquement accessibles au professeur. Ils déterminent

- la liste initiale des dix résultats
- la valeur de μ
- le choix du type de dialogue : saisie de la réponse ou dialogue oui/non

7.9. Contrôle de la séquence des énoncés rentrés au clavier

Quand l'ordinateur affiche la liste des exercices "à travailler" ou "mes bases", il s'agit toujours d'exercices dont la structure n'est pas encore assez connue. Lorsqu'un énoncé est rentré au clavier, il est donc utile d'effectuer un contrôle pour tester si sa structure est déjà bien connue ou non. Si cette dernière est déjà bien maîtrisée , cela lui sera signalé.

8. Des choix

8.1. Au sujet des expressions rentrées au clavier

Seules les expressions ne faisant intervenir que des nombres entiers ou fractionnaires seront admises. Par exemple, 4*6+1/2 est admis pour nous alors que 4*6+0,5 sera rejeté.

De même, les réponses que l'élève introduit au clavier seront impérativement des entiers ou des fractions à l'exclusion de produits, sommes, ... On n'exige pas de l'élève qu'il introduise comme résultat une fraction réduite. On pourrait concevoir que l'élève introduise une expression plutôt qu'une valeur. Cette option permettrait de mieux connaître la démarche suivie par l'élève.

8.2. Le public

Les élèves depuis le début du secondaire jusqu'en quatrième année sont le public cible. C'est en effet à partir de la première année que l'on commence à voir les rudiments d'algèbre. La conception même du logiciel permet de l'utiliser à n'importe quel niveau sans adaptation puisque l'élève rentre au clavier l'expression qu'il souhaite.

8.3. Le matériel

Je possède un Macintosh et j'ai donc choisi de développer le didacticiel sur mon matériel. Il faut prendre la précaution de désactiver, si nécessaire, l'adressage 32 bits, car sinon le logiciel provoque une erreur système.

Rien n'empêche bien entendu de l'implémenter sur un PC avec Window.

9. Extensions possibles des fonctionnalités

9.1. Les puissances

Ce serait utile d'inclure la puissance en plus des quatre opérations de base. Le logiciel pourrait alors traiter des expressions encore bien plus complexes. Il n'y a pas de difficultés majeures pour implémenter cette opération supplémentaire. Le seul obstacle que l'on peut y voir est l'écriture à l'écran d'une puissance.

Le logiciel écrit (2+5/6)^3 et on aimerait voir $(2+5/6)^2$. Il faut donc traiter systématiquement l'expression pour la redessiner à l'écran.

9.2. Stocker les confusions

Une fois trouvée, une confusion pourrait être stockée dans le fichier de l'élève. On pourrait adopter la forme suivante :

confusion(ExpDonnée, ExpConfondue).

Ces confusions sont évidemment propres à chaque élève au même titre que les structures.

Cette collection peut contribuer à améliorer le logiciel sur deux plans :

- Faire afficher le liste des confusions par l'élève pour être consultée à tout moment.
- Affiner la recherche automatique de l'erreur. Cette liste de confusions peut servir à mieux localiser la source probable de l'erreur.

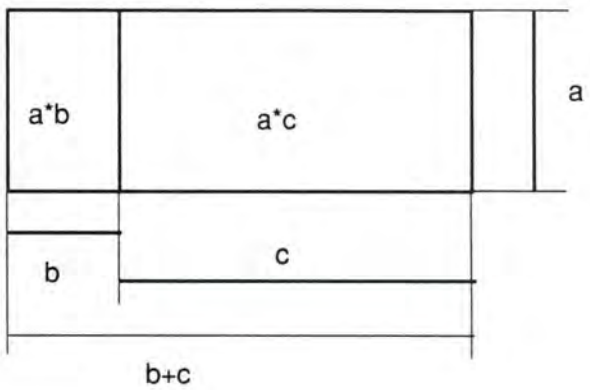
9.3. La sémantique des règles

Trop souvent une formule ne parle pas aux élèves. La difficulté pour retenir une formule réside souvent dans le fait qu'on n'y voit qu'une suite de symboles sans signification. Il n'est dès lors pas surprenant que la rétention de ces formules soit vue comme bien difficile et purement "débile". Autrement dit, ils n'en perçoivent pas toujours la sémantique. Pour faire "parler" une formule, on pourrait essayer de la présenter d'une autre manière en quittant le plan du symbole pour une représentation géométrique. L'élève pourrait ainsi "voir" la formule autrement.

Par exemple, la distribution peut être vue de deux façons :

- a*(b+c)=a*b+a*c
- et aussi comme ceci :

a*(b+c)=a*b+a*c



Ce n'est pas évident de trouver une telle modélisation pour les différentes expressions courantes.

Le schéma adéquat pourrait apparaître à l'écran en même temps qu'une formule est mentionnée dans les commentaires, ou à la demande de l'élève.

9.4. Les sélections multiples.

Le logiciel ne permet que de faire une sélection d'un seul tenant. Or il serait tout à fait indiqué de pouvoir faire une sélection multiple dans plus d'un cas; Par exemple, si on propose l'expression 1/2+4+5*(8-2)-6, il serait intéressant de pouvoir sélectionner 4 et -6. On ne regroupe pas toujours que des symboles contigus.

9.5. Réduire une expression.

Il s'agit ici de remplacer une sélection non pas par une valeur, mais par une expression équivalente. Par exemple,

2*(5+2)-7*2 devient 2*(5+2-7)

On déroge donc à la règle utilisée jusqu'ici et qui veut que l'on ne remplace une expression que par un entier ou une fraction d'entiers.

9.6. Présenter des exercices non choisis par l'élève

La liberté laissée à l'élève doit aussi s'accompagner de contraintes dictées par le professeur. C'est alors une phase d'évaluation "certificative". Le moment où des exercices non choisis par l'élève lui seraient proposés ou imposés serait décidé par le professeur. Tout les paramétrages sont envisageables.

- Le moment :
  - soit au hasard
  - soit après un certain nombre d'exercices.
  - soit après un changement d'un drapeau par le professeur et donc au début de la session suivante pour l'élève.

- Le caractère obligatoire.

On peut rendre ces exercices

- vraiment incontournables
- facultatifs.

La mémorisation de la réaction de l'élève à ces exercices peut être traitée à part et constituer une indication intéressante pour le professeur.

9.7. Corrections en chaîne.

Quand l'élève a choisi de décomposer lui-même une expression, la liste des réductions successives est affichée en permanence à l'écran.

Admettons que le logiciel accepte de réduire une expression avec une erreur dans l'évaluation de l'élève. Alors on peut concevoir que l'on puisse faire une correction dans la seconde expression de la liste et que cette correction se répercute dans les expressions suivantes. Nous aurions donc affaire à une possibilité de voir les conséquences d'une faute réalisée à n'importe quelle étape.

9.8. Accès aux paramètres.

Un système de mots de passe devrait protéger l'accès aux paramètres afin que le professeur soit le seul à pouvoir les modifier.

9.9. Des librairies

En fait on pourrait mettre à la disposition des élèves des librairies contenant des fichiers au contenu bien précis et mieux ciblé. On songe par exemple à un fichier axé sur la manipulation des quotients, un autre où on exclut tout quotient et où on se centre sur les distributions , ...

9.10. Les fonctions (utilitaires) du professeur

Énumérons seulement quelques idées :

- statistique sur le taux de réussites et d'échecs sur une session.
- statistique sur le taux de refus et d'acceptation des exercices proposés.
- mémorisation périodique de la connaissance d'une structure et voir son évolution dans le temps. Ces différentes valeurs peuvent être mémorisées dans une liste attachée à la structure concernée.
- réfléchir à la construction d'une connaissance globale obtenue par agrégation de certaines connaissances des structures ou créer des connaissances globales attachées à certains objectifs

Ces résultats peuvent être utilisés comme moyens de guidance pour le choix des exercices à proposer.

10. En résumé

Suite aux constats de la première partie, nous décidons de construire un logiciel qui essaye de situer l'erreur, de détecter son type, de mémoriser les expressions mal maîtrisées. Il faudra mémoriser les informations pertinentes.

Les informations qui semblent adéquates sont pour l'essentiel, la connaissance et la *structure* d'une expression. D'autres informations statistiques sont aussi intéressantes à retenir.

Les exercices que l'ordinateur va générer seront construis en fonction des réactions de l'élève aux exercices précédents et uniquement en fonction de ses réactions. Ils seront ainsi adaptés pour l'élève concerné.

Le cheminement vers l'abstraction aidera l'élève à évoluer vers une connaissance profonde des mécanismes mis en jeu.

L'affichage de commentaires pertinents est capital.

Troisième partie

Analyse fonctionnelle

1. Objectif

L'analyse fonctionnelle à pour but de décrire et de détailler les fonctions que le logiciel doit réaliser et ainsi de fournir une idée claire de ses possibilités. La façon dont ces fonctionnalités ont été réalisées ne fait pas partie de ce chapitre mais de l'implémentation.

2. Un exercice

Le logiciel propose aux élèves de faire des exercices.

Un exercice consiste à évaluer une expression numérique d'une complexité quelconque faisant intervenir les quatre opérations. Dans le cas d'une décomposition par l'ordinateur, on attend de l'élève qu'il donne la valeur de l'expression ou qu'il signale par oui ou non s'il a bien obtenu la bonne réponse qui est proposée. Dans le cas où la décomposition est manuelle, on attend la même chose, mais il faut préalablement sélectionner l'expression à évaluer.

Une expression est donnée telle qu'elle serait écrite au tableau, sauf que les quotients seront toujours sous la forme a/b . Tout est donc écrit sur une seule ligne.

Cette expression apparaît dans une fenêtre qui contiendra les commentaires et les autres exercices de toute une session (voir plus loin).

3. Les grandes fonctions du logiciel

Nous distinguons deux grandes fonctions.

- analyse des exercices;
- création d'exercices propres à l'élève.

Nous avons aussi, ce que l'on peut voir comme des fonctionnalités "associées" aux fonctionnalités de base :

- gestion de la base
- consultation des connaissances.
- gestion de l'interface
- affichage des commentaires

Ces fonctions doivent évidemment s'éclater en sous fonctions. ce point sera abordé plus loin.

4. Analyse d'un exercice

Cette fonctionnalité (ANALYSE)

reçoit

- une expression numérique
- les valeurs des paramètres qui détermineront le type de *décomposition* et de *recherche*.

après analyse, elle donne

- des commentaires
- met à jour la base de données

Elle est schématisée à la figure 3.2 p 40.

Faire des commentaires consistera à aider l'élève à la compréhension de ses erreurs.
La mise à jour de la base de données est faites par l'appel à la fonction Gestion de structures (voir plus loin).

Les commentaires varient suivant le mode de décomposition adopté pour l'expression. Ce détail sera donc donné plus loin dans un paragraphe réservé à cet effet et fera l'objet d'une fonctionnalité particulière.

La mise à jour de la base de données consiste à mettre à jour les "formes" (Voir seconde partie). Il faudra faire appel à des données temporaires, c'est-à-dire dont la durée de vie est celle d'un exercice, qui elles aussi doivent être mises à jour. Mais ces données temporaires dépendent de l'implémentation et ne seront donc pas mentionnées ici. Gérer les formes consiste à

- modifier une *forme*, c'est-à-dire modifier ses composants.
- créer une nouvelle *forme* dont les composants sont initialisés.

5. L'interface

L'interface doit être agréable et sobre. Il est essentiellement composé de quatre parties :

1 Les menus déroulants.
2 Une fenêtre Historique.
3 Une fenêtre Evolution.
4 Une boite de dialogue.

Voyons maintenant la fonction de chacun de ces éléments.

Les menus déroulants

Ils permettent

de faire les choix de départ :

- la fixation des paramètres pour la *recherche* et la *décomposition*,

de faire une série de consultations :

- la consultation d'un rapport, du *noyau* (voir plus loin) , de la liste des formules,

de faire des actions

- lancer un exercice d'un type déterminé
- changer d'utilisateur, quitter le logiciel

La fenêtre Historique

Son rôle est capital. Elle a pour but d'afficher le déroulement complet des exercices. Nous y trouverons donc

- l'énoncé
- les commentaires sur le résultat
- les commentaires sur des confusions éventuelles
- les évaluations successives si l'élève sélectionne lui-même les expressions qu'il va évaluer

Tant que l'utilisateur ne change pas, cette fenêtre n'est pas effacée. A la fin de sa session, l'élève pourrait donc imprimer le contenu et conserver une information détaillée de son travail.

La fenêtre Evolution

Elle indique aussi un historique, mais relatif à un exercice et non une session.
Elle mentionne :

- l'énoncé
- la liste chronologique des sous-expressions évaluées. Cette liste s'allonge donc dynamiquement

Lorsqu'un nouvel exercice est lancé, cette fenêtre est nettoyée.

La boite de dialogue

Elle permet de

- saisir la réponse de l'élève
- afficher des messages locaux ("erreur de syntaxe", "rien n'est sélectionné", ..)

Les deux fenêtres sont fixes durant toute une session alors que le dialogue est intermittent. Cet interface a des limitations liées au logiciel utilisé. Le caractère intermittent du dialogue en est une.

6. Création d'exercices.

Cette fonctionnalité a pour but de donner à l'élève les moyens de réaliser un exercice. En se basant sur les considérations développées dans la seconde partie, nous mettons à sa disposition cinq possibilités :

1 rentrer un énoncé au clavier
2 afficher la liste des *structures* garnie d'un entier aléatoire et en choisir une
3 afficher la liste des énoncés préparés par le professeur et en choisir un
4 afficher la liste des *familles* (énoncé littéral) et en choisir une. Ici l'énoncé sera dégarni de ses lettres et regarni avec des entiers
5 afficher la liste des *noyaux* et en choisir un. Il subit le même traitement que l'énoncé d'une famille.

La tâche essentielle de la fonctionnalité sera donc la construction de listes **et** le choix d'un élément de cette liste sauf pour le premier choix où elle effectue un contrôle syntaxique.

7. Eclatement d'une fonctionnalité

7.1. Avertissement

Après avoir décrit les fonctionnalités, on les éclate pour diverses raisons dont la plus importante est de rendre le travail réalisable. On peut situer cette étape dans la modularisation puisque celle-ci est une conséquence de l'éclatement des fonctions de départ. Nous avons préféré les situer ici car on les voit sous la forme d'une analyse plus fine des fonctionnalités.

7.2. L'approche orientée "fonction"

Elle est basée sur une découpe en termes de fonctions et compositions de fonctions. Il s'agit alors d'un diagramme de flux de données. Un diagramme (voir figure 3.1) a comme composant

- arcs : ce sont les Flux de données. L'orientation d'un arc indique le sens dans lequel voyage l'information
- rectangle : qui représente la base de données
- cercles : qui représentent les transformations

Les arcs orientés indiquent le sens dans lequel l'information circule.

Par convention :

- Le signe * entre deux arcs signifie qu'une donnée ne peut aller sans l'autre.
- Un segment entre deux ou plusieurs arcs signifie qu'ils s'excluent mutuellement.
- L'absence de * ou de segment entre deux arcs signifie l'absence de lien entre les deux arcs.

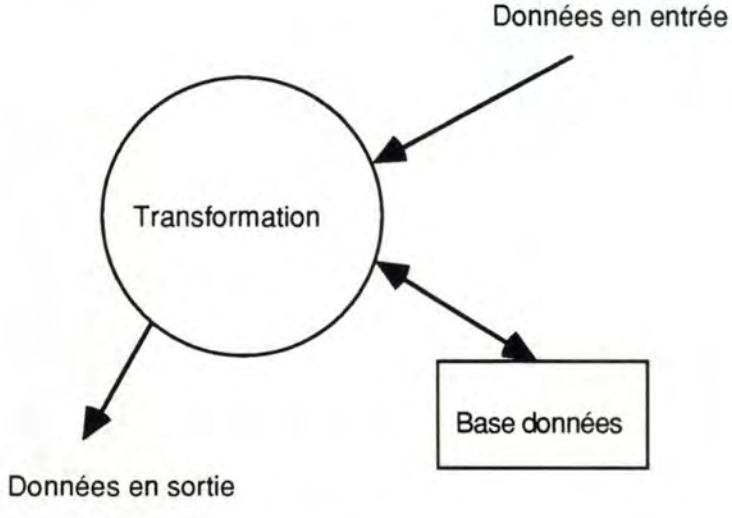


Figure 3.1

Approche orientée "fonction"

La démarche consiste donc à

- Identifier les flux de données en entrée et en sortie.

- Déterminer un certain nombre de transformations pour définir la relation entre flux de données en entrée et en sortie.
- définir ces transformations.

7.3. Les affinements

Notons que, lorsqu'on affine une fonctionnalité, il faut tester que l'on retrouve les mêmes données en entrée et en sortie.

7.4. Navigation

Pour que le lecteur puisse s'y retrouver sans difficulté j'ai présenté ci-dessous la vue générale de l'éclatement en sous fonctionnalités

Analyse d'un exercice

Extraction d'une sous expression
    Localisation automatique
    Type de décomposition suivant la valeur des paramètres
    Parcourir arbre
    Evaluation de la sélection
        Sélection d'une sous expression
        Syntaxe de la sélection
        Sémantique de la sélection
        Affichage d'un message d'erreur et resélection
    Construire l'expression réduite

Traiter la réponse de l'élève
    Détermination du type de dialogue
    Saisir la réponse de l'élève
        Syntaxe de la réponse
        Vérification de la réponse
    Dialogue type oui/non

Recherche d'une éventuelle confusion
    Recherche si la structure est à risque
    Correspondance entre la valeur de la réponse et une fausse règle

Affichage des commentaires

Création d'un exercice

Détermination de l'origine
Création d'un exercice personnel
    saisir un énoncé au clavier
    vérification syntaxique
    affichage de l'erreur et saisie d'un nouvel énoncé
Appel des exercices du professeur et choix de l'un d'eux
Création d'un exercice de la liste des structures
Création d'un exercice de la liste des familles

Nouvel utilisateur

Saisie de l'identité de l'utilisateur
Ouverture et chargement du fichier

7.5. Analyse d'un exercice

Cette fonction reprend (Voir figure 3.2) la description faites plus haut.

- En entrée : l'exercice **et** les valeurs des paramètres sur le type de *décomposition* et sur le type de *recherche*.
- En sortie : la base est mise à jour et les commentaires sont affichés.

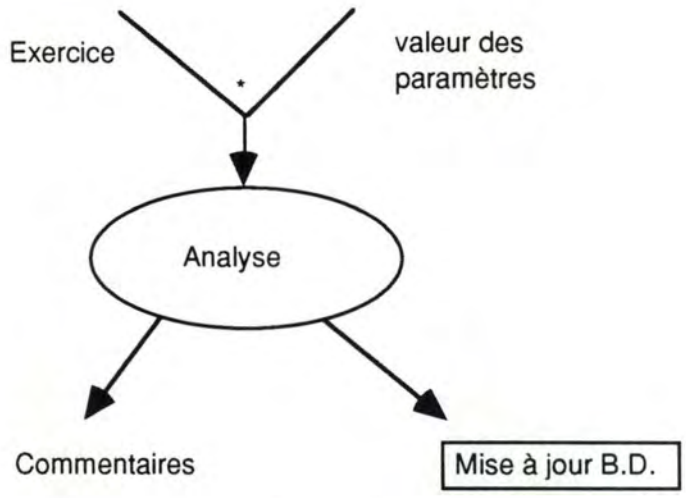


Figure 3.2

Fonctionnalité ANALYSE

7.6. Création d'un exercice

Cette fonctionnalité est représentée figure 3.3.
En entrée :
une demande d'énoncé
En sortie :
une expression valide qui sera passée à la fonctionnalité Analyse.

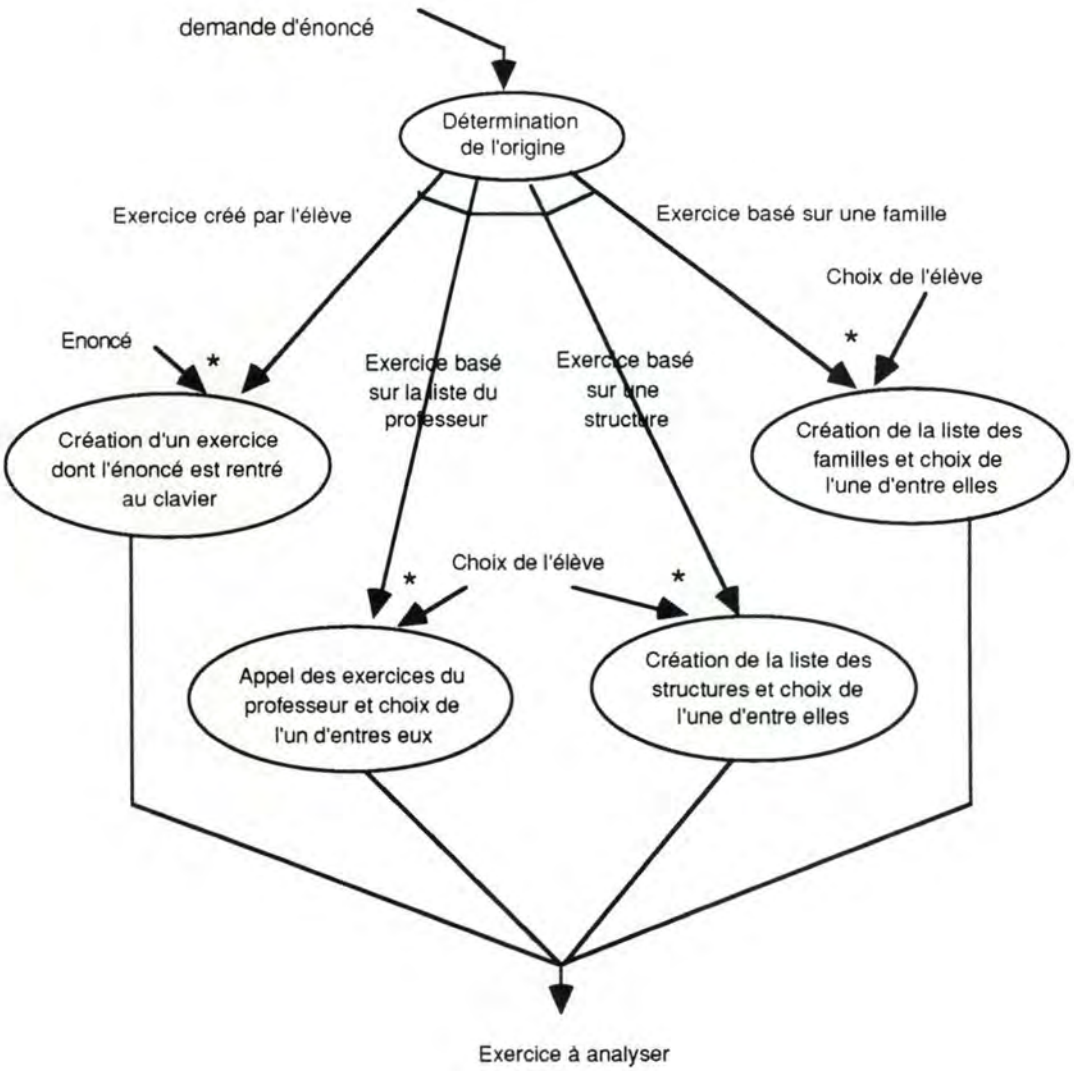


Figure 3.3

La fonctionnalité CREATION D'UN EXERCICE

7.7. Eclatement de Création d'un exercice personnel

Cette fonctionnalité est représentée à la figure 3.4.
En entrée :
la demande d'un exercice dont l'énoncé est rentré au clavier par l'élève.
En sortie :
une expression dont la validité a été vérifiée.

On constate ici qu'il faut prévoir le cas où la syntaxe est incorrecte avec la possibilité de réintroduire un nouvel énoncé dont la syntaxe sera revérifiée.

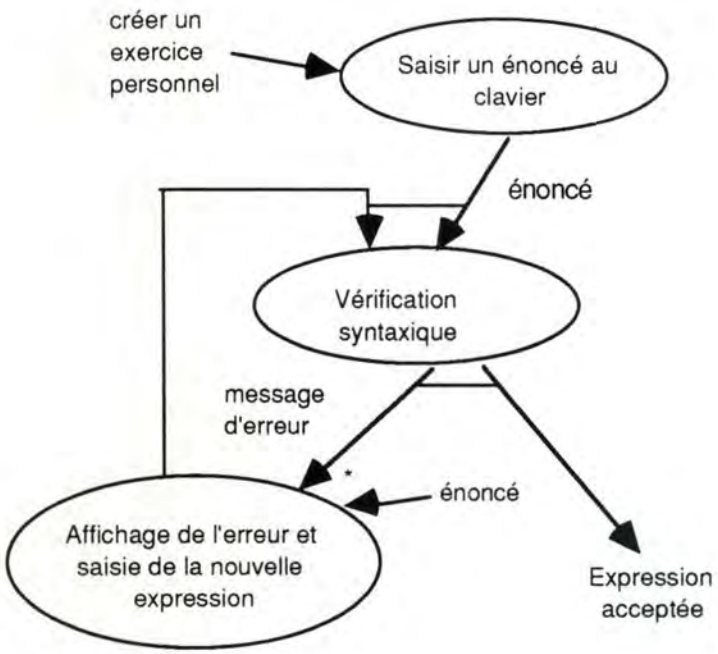


Figure 3.4

Eclatement de Création d'un exercice personnel

7.8. Syntaxe

La vérification syntaxique revêt deux aspects.

D'une part la vérification de la syntaxe proprement dite (l'expression est-elle bien construite ?). Ensuite la vérification du type de l'expression. En effet, la structure attendue de l'expression dépend de l'expression considérée. S'il s'agit de la valeur d'une expression, on veut avoir affaire uniquement à un entier, ou à une fraction. Par contre, s'il s'agit de l'énoncé d'une expression, la structure attendue est bien plus large comme nous le détaillons plus bas.

On souhaite aussi que l'endroit où on rencontre une erreur pour la première fois soit signalé à l'utilisateur. Supposons que l'élève entre l'expression suivante

5 * ((9 - / 2)

On y constate deux erreurs. Quand on parcourt l'expression de gauche à droite, la première erreur rencontrée est une parenthèse ouvrante inutile. C'est à cet endroit qu'il faut attirer l'attention de l'élève. On aura, par exemple, un affichage comme suit :

5 * (9 - # / 2))

où le symbole # indique qu'à partir de cet endroit il y a un problème.

7.8.1. Validation de la syntaxe d'un énoncé

Une expression est définie récursivement et doit être de la forme décrite à la figure 3.5, Elle signifie qu'une expression est un terme éventuellement suivi d'autres termes tous séparés par un opérateur additif. L'expression peut débuter par le signe "-".

Un terme est (voir figure 3.6) un facteur éventuellement suivi d'autres facteurs séparés par un opérateur multiplicatif.

Un facteur est (voir figure 3.7) un nombre ou une expression.

Un nombre est (voir figure 3.8) un chiffre éventuellement suivi d'autres chiffres. Un nombre peut débuter par un signe "-".

7.8.2. Validation de la syntaxe de la valeur d'une expression

La figure 3.8 donne la forme attendue d'une valeur. C'est un nombre éventuellement divisé par un autre nombre.

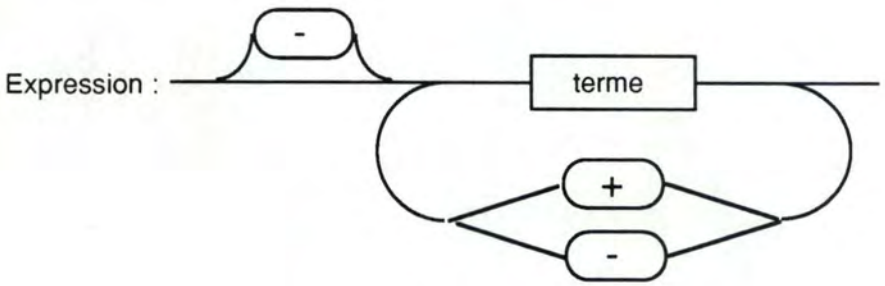


Figure 3.5

Syntaxe d'une expression

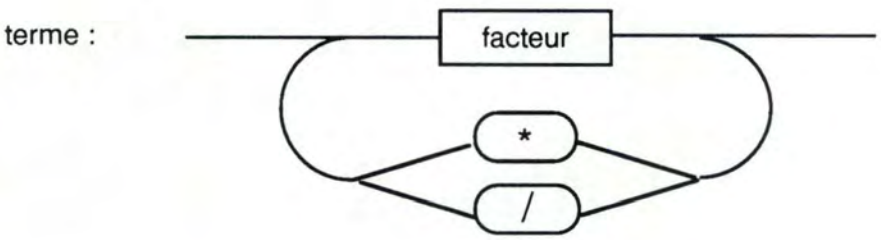


Figure 3.5

Syntaxe d'un terme

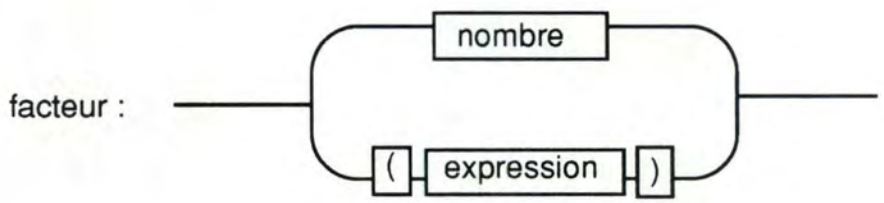


Figure 3.6

Syntaxe d'un FACTEUR

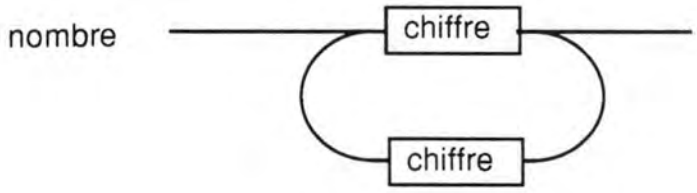


Figure 3.7

Syntaxe d'un NOMBRE

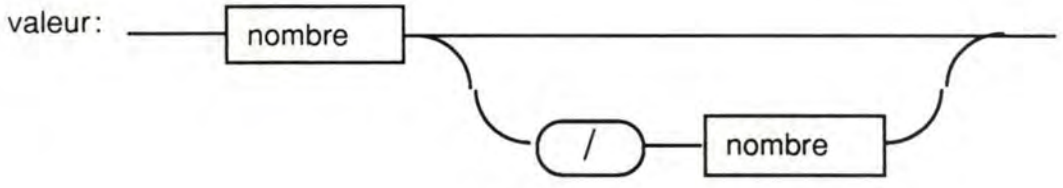


Figure 3.8

Syntaxe de la réponse d'un élève

Notons que les expressions du type 5+-6 sont bien rejetées et qu'un signe "-" est admis au début d'une expression. On peut donc avoir 5*(-6+5).

7.9. Eclatement d'Analyse

Cette fonctionnalité se décompose (voir figure 3.9) en trois nouvelles fonctionnalités :

- Extraction d'une sous expression. Elle décompose l'expression, soit à l'aide de l'ordinateur, soit manuellement.
- "Traiter la réponse de l'élève". Cela consiste à vérifier la syntaxe de la réponse de l'élève et ensuite à contrôler si cette réponse est correcte.
- "Recherche d'une éventuelle confusion". Cette recherche consiste à vérifier si une fausse règle n'a pas été appliquée.
- "Afficher les commentaires". Elle affiche les commentaires en consultant la base de données où se trouvent toutes les informations nécessaires qui ont été stockées durant la décomposition.

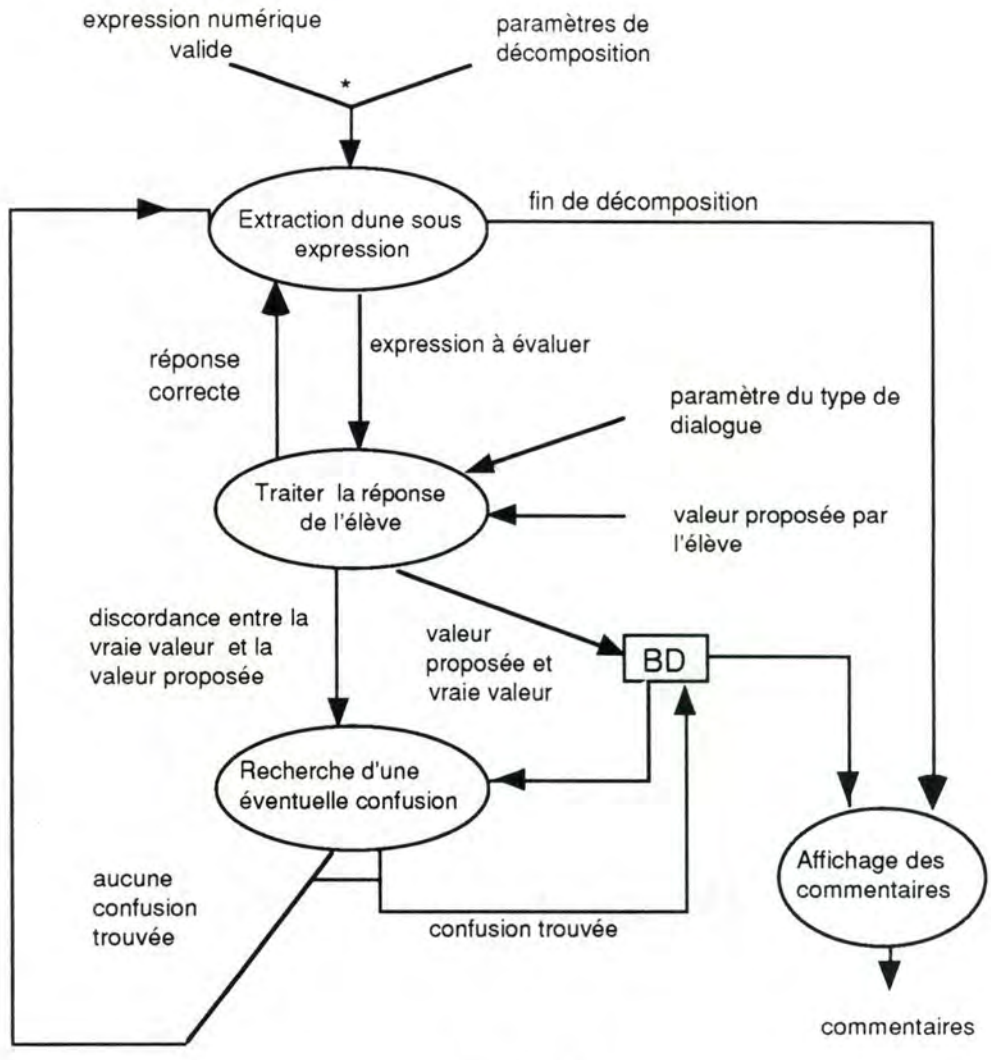


Figure 3.9

Eclatement de la fonctionnalité ANALYSE

Si l'expression de départ est mal évaluée, alors une sous-expression en est extraite par une des méthodes vues plus haut. Cette sous expression est ensuite passée à la fonctionnalité chargée de saisir la réponse pour l'évaluer et comparer sa valeur à celle proposée par l'élève. S'il y a une différence entre ces deux valeurs, ces deux valeurs sont passées à la fonctionnalité chargée de détecter l'application d'une fausse règle éventuelle. La base à été continuellement mise à jour durant ces opérations. Ensuite les commentaires sont affichés en consultant la base pour les réaliser.

7.10. Eclatement de "Extraction d'une sous expression"

La fonctionnalité "Extraction d'une sous expression" se scinde en cinq nouvelles fonctionnalités (voir figure 3.10):

- localisation automatique
- parcourir arbre
- évaluer sélection
- construire expression réduite
- valeur du paramètre

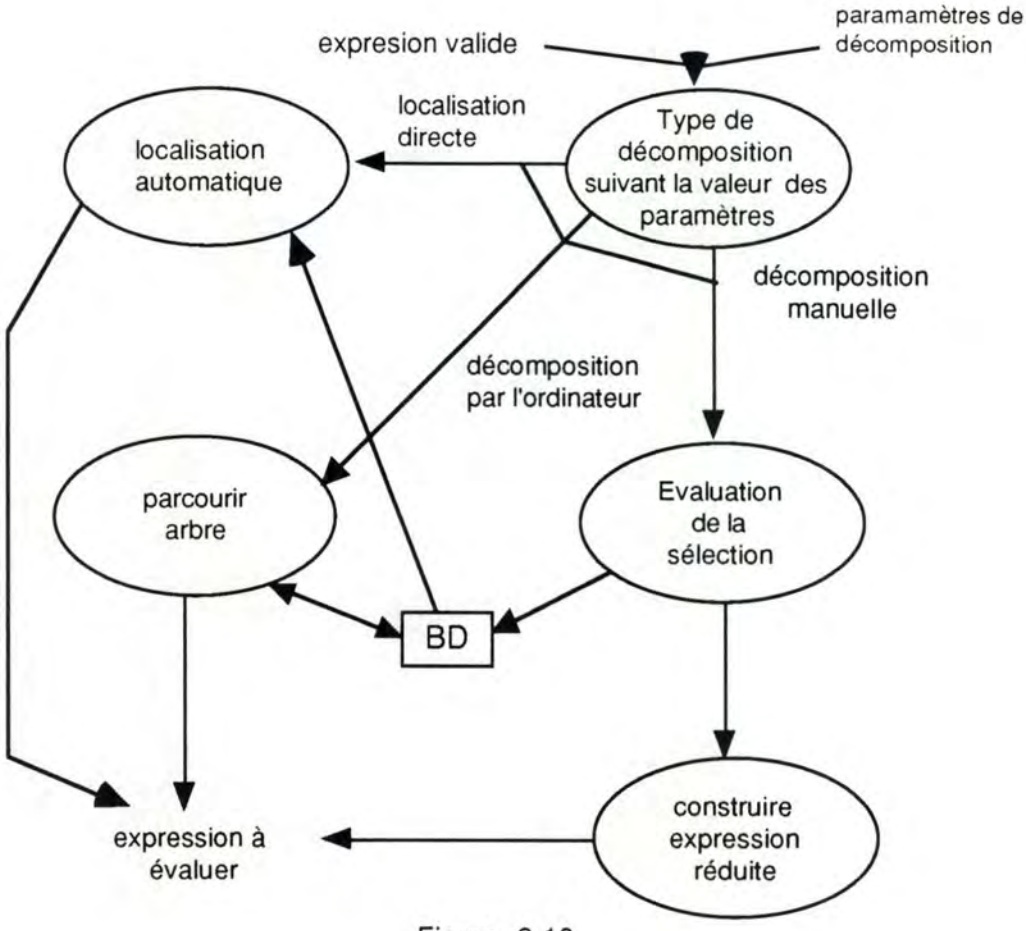


Figure 3.10

Eclatement de EXTRACTION D'UNE SOUS EXPRESSION

Suivant la valeur du paramètre, une des trois démarches vues est adoptée :

- recherche automatique
- recherche interactive
- décomposition manuelle

7.11. Eclatement de Traiter la réponse de l'élève

La figure 3.11 illustre cette fonctionnalité. Suivant la valeur du paramètre, on active une des fonctionnalités "Saisir la réponse de l'élève" ou "Dialogue du type oui/non" Dans chaque cas la base de donnée est mise à jour.

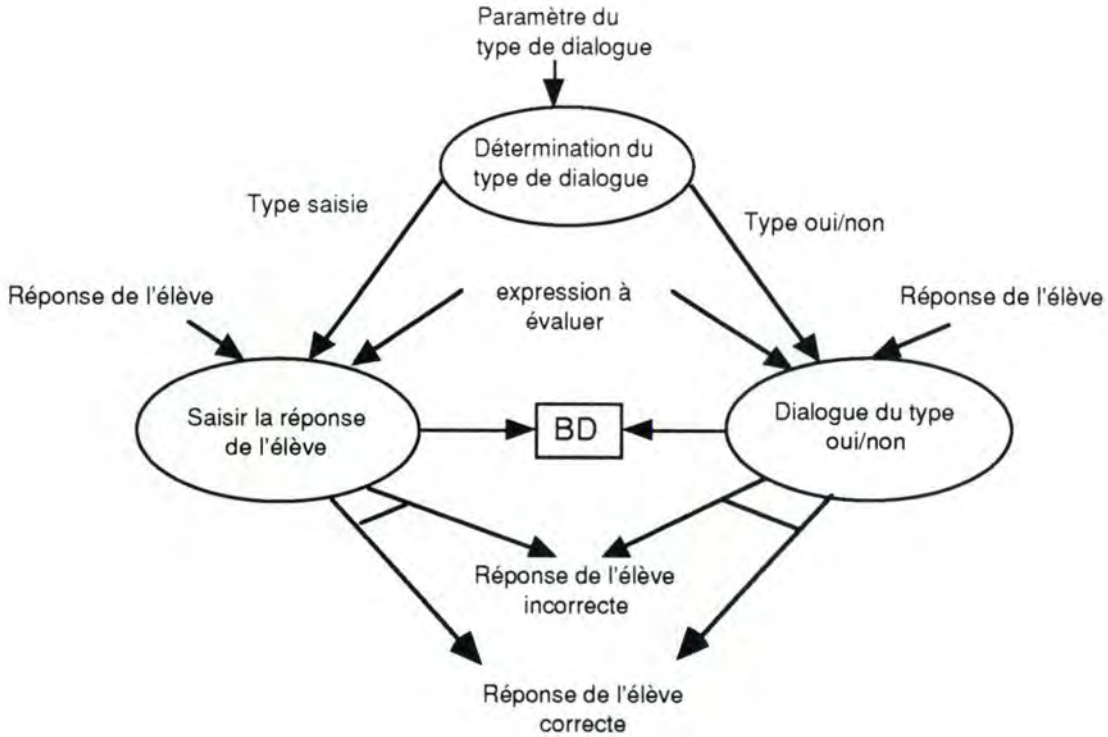


Figure 3.11

Eclatement de TRAITER LA REPONSE DE L'ELEVE

7.12. Eclatement de Confusion

Cette fonctionnalité est représentée à la figure 3.12. "Correspondance entre la réponse de l'élève et une fausse règle" recherche les confusions habituellement rencontrées dans l'enseignement. Rappelons que la liste de ces confusions est dressée par l'enseignant.

L'expression à évaluer ainsi que la réponse proposée par l'élève sont dans la base de données

7.12.1. Orienter la recherche.

On peut chercher de manière très générale quelle est la structure qui, avec les nombres intervenant dans l'expression, donne le résultat proposé par l'élève. Le hic, c'est que plus d'une fois cela conduit à une absurdité au niveau pédagogique. Voyons deux situations :

Exemple 1.

L'expression à évaluer est (2+4)*6. L'élève propose au hasard ou par erreur de frappe le nombre 12. L'ordinateur proposera (6*4)/2, faisant ainsi intervenir un opérateur qui n'est pas dans l'expression de départ.

Exemple 2

L'expression à évaluer est 2*(5+3). L'élève propose le nombre 13. L'ordinateur peut proposer 5*3-2, faisant ainsi intervenir un opérateur qui n'est pas dans l'expression de départ. Ce n'est pas le genre d'erreur que l'on rencontre dans la pratique.

Le principe de recherche adopté est le suivant :

7.12.2. La structure est-elle à risque ?

On a une expression pour laquelle l'élève propose une réponse incorrecte. On détecte, pour cette expression, si sa structure fait partie d'une liste de structures dont on sait, par la pratique, qu'elle génère facilement des confusions.

7.12.3. L'une des fausses règles pourrait-elle avoir été appliquée ?

La réponse proposée par l'élève permet alors de proposer une fausse règle comme source de l'erreur.

7.12.4. L'élève a t il bien fait cette confusion ?

Le dialogue propose la confusion à l'élève qui dit si c'est bien le cas. Si oui, une confusion est mémorisée et signalée dans la fenêtre Historique.

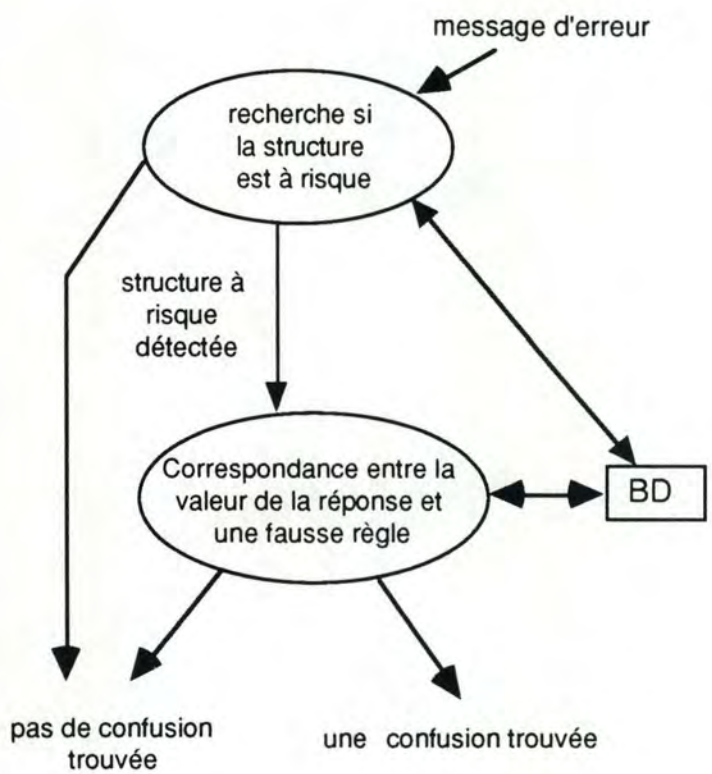


Figure 3.12

Fonctionnalité Recherche d'une confusion

7.13. Eclatement de Saisir la réponse de l'élève

Saisir une réponse comporte deux étapes : la vérification de la syntaxe puis, lorsque cette syntaxe est correcte, la comparaison de cette valeur à la vraie valeur de l'expression (voir la figure 7).

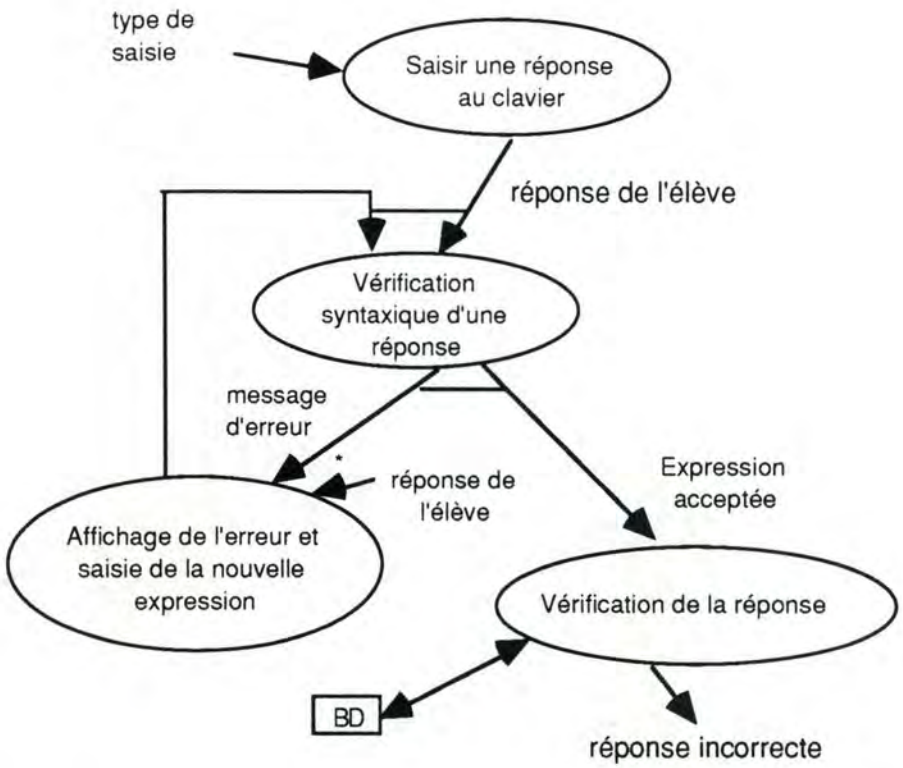


Figure 3.13

Eclatement de Saisir la réponse de l'élève

Cet éclatement est fort proche de celui de Création d'un exercice personnel mais il est différent. En effet, la vérification syntaxique n'est pas la même car on attend ici un nombre entier ou un nombre fractionnaire et il y a une fonctionnalité supplémentaire qui est la vérification de la réponse.

7.14. Eclatement de Evaluation de la sélection

Il faut distinguer deux cas.

Le premier est celui où la décomposition est effectuée par l'ordinateur. Dans ce cas on a l'assurance que la sous expression qu'il va falloir considérer a une sémantique correcte. Ce n'est pas le cas lorsque l'élève choisit sa sous-expression. Voyons cela de plus près.

7.14.1. Validation d'une sélection

Remarque : on ne traite ici que d'une sélection d'un seul tenant.

La validation d'une sélection soulève deux problèmes :

- la vérification syntaxique.
- la validité de la sélection même si elle est syntaxiquement correcte.

Illustrons le problème : soit l'expression

2+5*(3+4)

Si on sélectionne *(3+4) la syntaxe n'est pas correcte.

Si on sélectionne 2+5 la syntaxe de la sélection est correcte, mais la sélection ne l'est pas puisque l'on ne peut pas associer ces deux termes. Savoir si une sélection est valide n'est pas élémentaire. Concevons une expression comme un arbre. Alors la sélection d'un sous arbre est correcte, mais toute sélection correcte n'est pas nécessairement un sous arbre de l'arbre de départ.

Par exemple, dans l'expression 1+2-3+4+5 on peut associer 2-3+4 qui n'est pourtant pas un sous arbre de l'arbre de départ. Il faudrait donc pouvoir envisager TOUS les arbres équivalants à celui de départ et rechercher si 2-3+4 n'est pas un sous arbre d'un de ces arbres-là. Ceci semble fort lourd.

On peut cependant dégager une condition suffisante pour vérifier qu'une sélection n'est pas valide.

Soit l'expression E1 dont une partie S a été sélectionnée.

Si en remplaçant dans E1 la partie S par sa valeur, la nouvelle expression obtenue E2 a une valeur différente de celle de E1, alors la sélection n'est pas valide.

La condition n'est pas nécessaire. Quels sont les cas qui nous échappent ?

Par exemple, on a+b*c=(a+b)*c uniquement lorsque a=0 ou c=1.Il suffit de tester sur 1+1*1 où la sélection de 1+1 conduit à la bonne réponse. On peut ainsi relever pour chaque structure simple la liste des cas qui échappent au contrôle. En pratique, rares sont donc les sélections incorrectes qui ne sont pas détectées.

Ce test, dont on est conscient des faiblesses, permet une implémentation rapide et est une solution d'attente mais finalement assez efficace.

Le remplacement de la sélection par sa valeur doit maintenir la validité syntaxique de la nouvelle expression. Par exemple, lorsqu'une expression est correctement sélectionnée, sa valeur peut être négative. Il faut alors introduire des parenthèses pour éviter des situations du type où 2+(5-7)*2 devient, après sélection de (5-7), 2+-2*2 dont la syntaxe n'est plus correcte.

La fonctionnalité Evaluation de la sélection peut alors se décomposer comme indiqué à la figure 3.14.

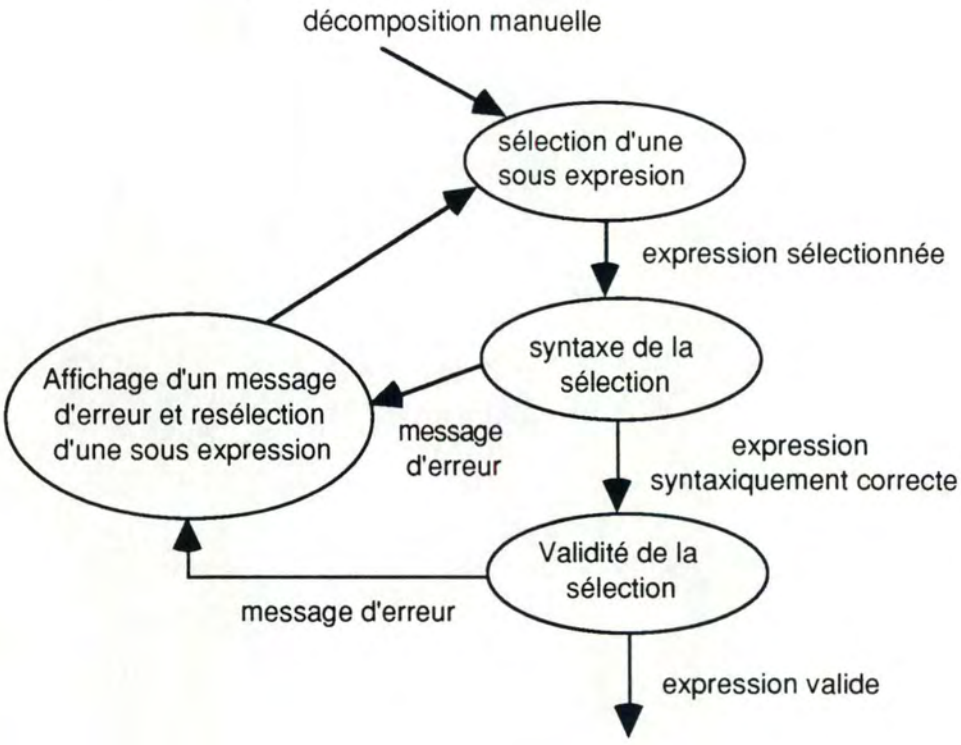


Figure 3.14

Eclatement d'évaluation de la sélection

7.15. Nouvel utilisateur

Cette fonctionnalité est appelée à l'ouverture du logiciel et chaque fois que l'on changera d'utilisateur.

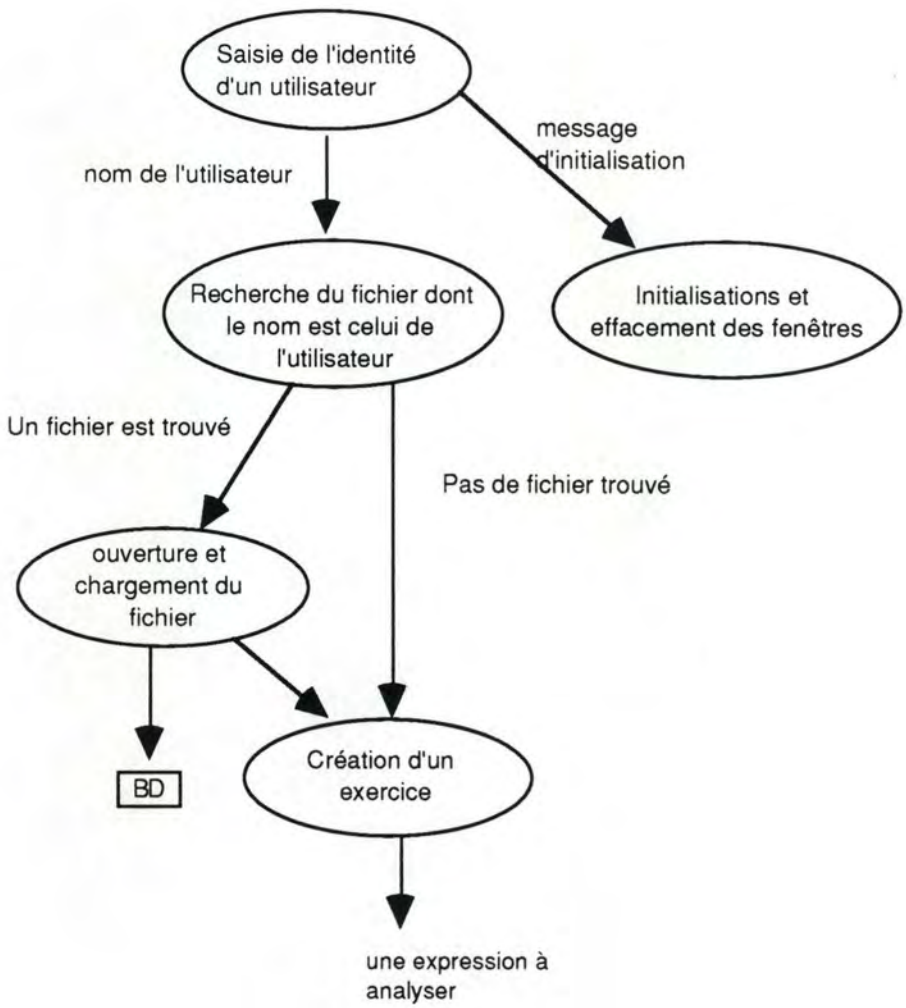


Figure 3.15

Fonctionnalité Nouvel utilisateur

7.16. La gestion des structures.

Rappelons qu'une structure est une expression dépouillée de ses nombres et dont on ne conserve donc que l'architecture. On y attache toute une série de renseignements.

Gérer ces structures consistera donc en :

- ajout d'une nouvelle structure différente de celles déjà enregistrées.
- modification des renseignements attachés à une structure, à savoir
  - mise à jour des différents compteurs.
  - mise à jour de la provenance de la structure.
  - mise à jour de la connaissance d'une structure.
  - mise à jour de la liste des dix derniers résultats.

- suppression des structures quand on change d'utilisateur.

- tri des structures suivant la valeur de la connaissance et d'un ou plusieurs critères.
    - pour afficher la liste des faiblesses.
    - pour afficher la liste des exercices du prof non encore réalisés par l'élève.
    - pour réaliser un rapport général avec une appréciation.
    - pour un essai de recherche automatique.

Notons qu'il n'y a pas de suppression de structures pour un élève donné.

Un mot au sujet de la liste des faiblesses.
Ce qui apparaît à l'écran n'est pas une liste de structures, mais une liste de structures garnies de nombres entiers. Ces nombres sont générés aléatoirement. De plus, ces structures garnies sont précédées de la valeur de la connaissance de chacune des structures garnies. Ceci peut permettre un choix plus circonstancié.

7.17. Gestion des fichiers

Cette fonction n'est appelée que dans cinq circonstances :
- quand on quitte le logiciel
- quand on change d'utilisateur
- quand on ouvre le logiciel pour la première fois.
- quand on a changé la valeur d'un paramètre.
- quand on charge les exercices proposés par le prof et stockés dans un fichier ad hoc.

Toutes les structures d'un élève et les renseignements qui y sont attachés formeront les données permanentes de cet élève. Il s'agira donc de :

- charger le fichier d'un utilisateur déjà existant. Les structures chargées constitueront donc les données disponibles sous forme de faits.
- créer un nouveau fichier du nom de l'utilisateur, qui va, le cas échéant, écraser le fichier déjà existant.

Quand on a changé la valeur d'un paramètre, il faut sauver le nouveau texte de tout le fichier, sinon la nouvelle valeur est perdue.

7.18. Rapport général

Cette fonction a deux objectifs.

7.18.1. Évaluation de chacune des structures.

Il s'agit d'attacher un jugement à chacune des structures sans exception. Ce jugement se veut une indication sur la connaissance de l'élève relative à la structure ainsi que son évolution. Les données sont :

- la valeur de la *connaissance*.
- la valeur des dix derniers résultats.
- le nombre de résultats corrects/incorrects pour cette structure. La somme des deux compteurs vus dans la *forme* relative à cette structure donne ce renseignement.

Il y a maintenant des choix arbitraires à faire sur les seuils qui seront fixés.

Rappelons que pour avoir une connaissance supérieure, par exemple, à 0,7, il faut deux choses :
- la structure doit avoir été rencontrée suffisamment de fois

- avoir été aussi reconnue comme maîtrisée dans la majorité de ces fois-à.

Appelons
- C la connaissance
- Cs le seuil de la connaissance à franchir pour avoir une "bonne" connaissance
- Cp la connaissance récente, c'est-à-dire calculée sur les D dernières expériences
- e l'écart jugé significatif entre Cp et C pour déclarer un progrès
- N le nombre total de fois qu'est rencontrée une structure.
- Ns le seuil inférieur pour N qui est jugé nécessaire pour que l'information soit significative.

Les grandeurs Cs, D,e et Ns sont des paramètres à fixer.

Nous avons alors les cinq situations suivantes :

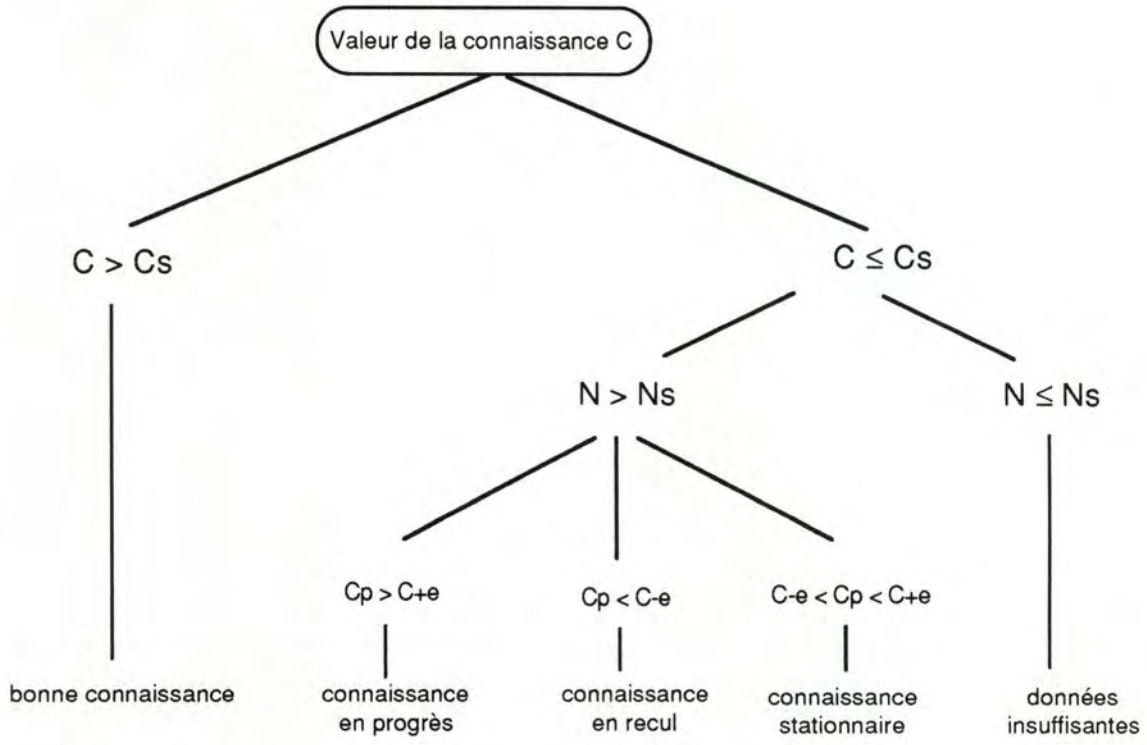


Figure 3.16

Les différents verdicts

Ensuite, les structures sont triées par type de verdict de manière à être présentées groupées.

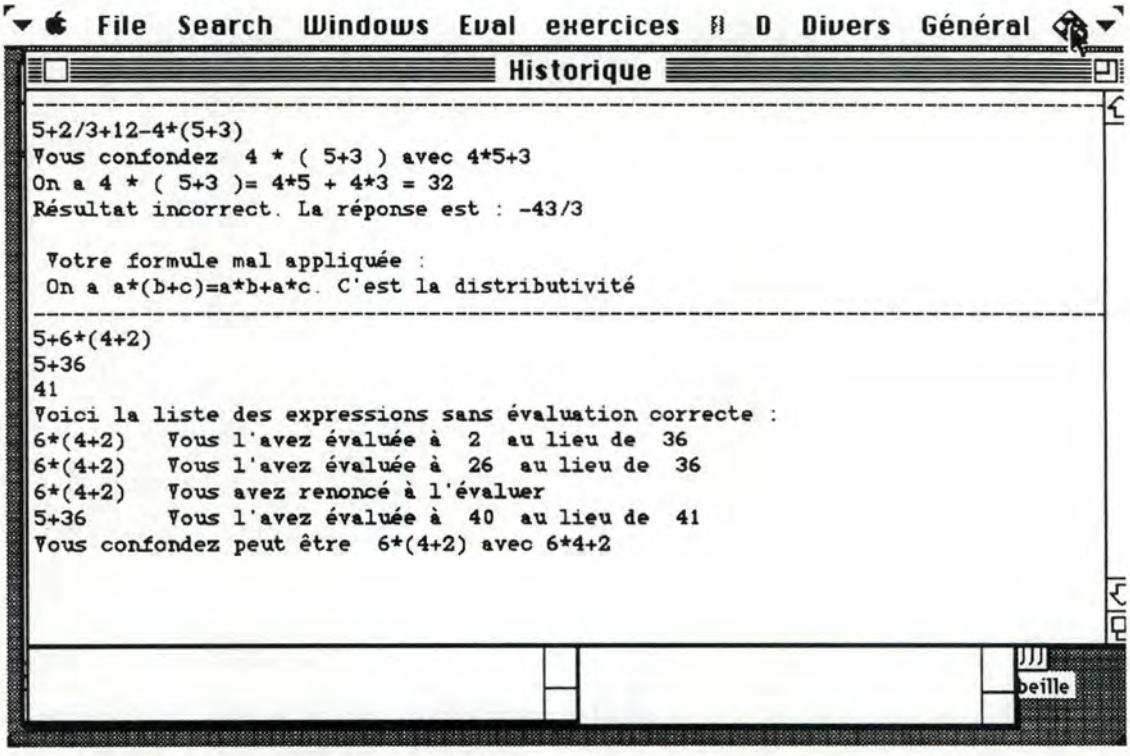
7.18.2. Évaluation de familles de structures

La liste des structures peut devenir très grande. Il est donc intéressant de réfléchir à un regroupement qui aurait du sens. Ce point est discuté dans "Vers l'abstraction".

Pour notre rapport nous sélectionnerons d'abord les *familles*. Avec cet ensemble comme point de départ, nous rechercherons ensuite le *noyau* des difficultés.

7.19. Les commentaires

Les commentaires sont réalisés lorsque l'exercice est terminé. Nous avons déjà traité de l'esprit de ces commentaires. Pour l'illustrer voici un exemple d'évolution de la fenêtre Historique. Le premier exercice a été décomposé par l'ordinateur. Le second a été décomposé par l'utilisateur. Pour ce dernier cas, on y voit l'énoncé et l'évolution de cette expression jusqu'à l'obtention d'un résultat qui est obligatoirement la bonne réponse.



Rappelons que dans le premier cas, on est *tout à fait certain* que l'élève a fait une erreur de distribution. En effet, la valeur proposée par l'élève pour l'évaluation de 4*(5+3) est 23 (ceci n'apparaît pas dans le commentaire mais on peut l'envisager) et ensuite il lui a été demandé s'il a bien fait cette confusion. Sa réponse a été positive.

Seule l'utilisation du didacticiel sur le terrain pourrait finalement dire si les commentaires sont pertinents. Peut-être faut-il en supprimer certains ou en ajouter d'autres.

Quatrième partie

Analyse organique

1. Objectif

Cette analyse a pour but de structurer l'ensemble des procédures du logiciel. Elle se situe après l'analyse fonctionnelle et avant l'implémentation. Il faut donc préalablement avoir éclaté les fonctionnalités en fonctionnalités plus fines.

Ces sous fonctions seront donc regroupées suivant certains critères. Un regroupement est un module. Ces modules communiquent entre eux. La représentation de cette communication sera vue plus loin.

2. Les modules.

2.1. Fonctions de la modularisation.

La modularisation a comme objectif de structurer l'ensemble du code pour en faciliter

- la réalisation.
- la réutilisabilité.
- la maintenance.
- la vérification par des tests.

Il s'agit d'une division du travail. où chaque module doit rester "maîtrisable".

Les principes de la réalisation de ce regroupement se fonde sur la notion d'interaction On appelle interaction la quantité d'informations échangées entre les fonctionnalités.

2.2. Les principes du regroupement.

Deux principes sont à la base du regroupement :

- interaction externe faible. Les échanges entre modules doivent être les plus réduits possibles et ce, pour atteindre les objectifs décri plus haut.
- interaction interne forte. Il faut que les échanges internes soient plus nombreux que les échanges avec l'extérieur sans qui on perd le bénéfice de la modularisation; En quelque sorte, on cache à ceux qui utilisent un module la façon dont on a réalisé les fonctionnalités disponibles du module.

Chaque module a ainsi un critère qui unit ses composantes. Les critères qui conduisent au regroupement sont multiples et laissés au choix du concepteur. Ils peuvent aussi être de nature différente d'un module à l'autre.

- La cohésion de type logique : par exemple les fonctions scientifiques.
- La cohésion de type objet : porte sur les même structures.

- La cohésion de type fonctionnel : ce sont les sous fonctions qui forment une fonction.

Mais notons qu'un module peut contenir autre chose que des foncions, par exemple des variables, des données ... (voir plus loin Stockage des paramètres).

2.3. La relation "Utilise".

Considérons deux modules A et B. Des fonctions de A peuvent utiliser des fonctions ou des éléments qui sont logés dans B. Nous matérialiserons ce fait par une flèche partant de A et dirigée vers B et qui se lit "utilise". Ainsi nous avons dans notre cas A -> B.

Quand on conçoit A on utilise B sans s'occuper de sa réalisation. Quand on conçoit B on ne s'occupe pas de A.

2.4. Les niveaux.

La modularisation conduit à une découpe du code en niveaux. Traditionnellement elle se réalise en cinq niveaux :

2.4.1. Niveau 5 : Les fonctionnalités

Il dérive directement des spécifications fonctionnelles.

2.4.2. Niveau 4 : Gestion des Entrées/Sorties de type externes.

Il se charge de l'aspect communication avec l'extérieur. Il s'agit notamment de la saise des données, des contrôles syntaxiques qui y sont liés, de la sémantique, des messages à afficher.

2.4.3. Niveau 3 : Gestion des Entrés/Sorties de type internes.

On s'occupe des données persistantes. Il s'agit de l'accès aux fichiers et aux bases de données.

2.4.4. Niveau 2 : Les outils du logiciel utilisé.

Le logiciel dont il est question est l'outil informatique utilisé. Ces informations sont prédéfinies et sont à la disposition de l'utilisateur. Le manuel qui accompagne le logiciel contient la liste exhaustive de ces informations.qui sont la totalité de l'environnement mis à notre disposition.

2.4.5. Niveau 1 : Le système d'exploitation de l'ordinateur.

Comme pour le niveau 2, ce sont évidemment des données prédéfinies.

3. Liste des modules

Il faut se référer à la figure 4.1 pour voir les liens entre modules. Les liens sont matérialisés par la relatiion "utilise".

Ci dessous la liste des modules avec la tâche qu'ils réalisent. Le détail du contenu se trouve dans le chapitre IMPLEMENTATION.

Nous ne dirons rien sur les niveaux 2 et 1.

3.1. Niveau 5

1. Action menus

Nous trouvons ici toutes les commandes déclenchées par la sélection d'un item d'un menu.

Il y a cinq menus : Les premier, quatrième et cinquième déclenchent des actions alors que les deux autres sont des choix de paramètres.

1 Le menu "Exercices" qui lance un exercice dont le type est déterminé par le choix de l'item
2 Recherche.
3 Décomposition.
4 Divers.
5 Général.

2. Rapport

réalise le rapport général sur un élève. La cohésion interne est séquentielle.

3. Aide

contient les formules fondamentales de l'algèbre.

4. Création des exercices :

regroupe toutes procédures qui créent des exercices.

5.Extraction de sous expressions :

regroupe toutes les procédures qui permettent d'extraire une sous expression d'une expression donnée comme énoncé.

6.recherche des confusions :

contient toutes les procédures qui gérent les confusions.

7.Utilitaires

collection de sous modules .

- Calcul :
  regroupe toutes les fonctions de calcul. Cohésion de type logique
- Paramètres :
  Gère la mise à jour des paramètres. Cohésion de type séquentiel
- Gestion des listes partielles
- Divers

3.2. Niveau 4 (E/S externes)

Les modules suivant font partie du niveau 4 :

1.Interface : Gère la totalité de l'interface à savoir
- les fenêtres
- les dialogues

  L'interface est scindé en deux.pour des raisons expliquées dans l'implémentation

2.Syntaxe : valide la syntaxe d'une expression

3.Affichage des commentaires : gère l'affichage des commentaires

3.3. Niveau 3 (E/S internes)

1. Gestion de fichier : gère les fichiers des élèves et du professeur.

2. Gestion des faits : gère toute la collection des faits

3. Stockage paramètres : contient uniquement les paramètres de façon à pouvoir les mettre à jour quand on change la configuration.

Figure 4.1 Modularisation

Cinquième partie

Implémentation

---

1. Objectif

L'implémentation a pour but de créer un programme exécutable. Nous allons ici parler du langage utilisé et documenter des procédures. Le paragraphe "Navigation" (voir plus loin) vous indique comment naviguer afin de situer un prédicat précis.

2. Le langage

2.1. Le choix et ses raisons

C'est le langage PROLOG qui a été utilisé. Il s'agit d'un langage de programmation logique. Clocksin et Mellish [8] définissent Prolog ainsi :

> "Prolog is a computer programming langage that is used for solving problems that involve *objects* and the *relationships* between objects."

De même Shapiro et Sterling dans The Art of Prolog [7] disent :

> "A logic program is a set of axioms, or rules, defining relationships between objects. A computation of a logic program is a deduction of consequences of the program. A program defines a set of consequences, which is its meaning; The art of logic programming is constructing concise and elegant programs that have the desired meaning."

Une différence fondamentale est qu'avec un langage procédural on programme en tenant obligatoirement compte du déroulement chronologique des instructions. Avec Prolog, la chronologie n'est pas omniprésente. Elle intervient uniquement quand c'est nécessaire, et dans ce cas, plus tardivement qu'avec un langage procédural.

Prolog se prête bien au prototypage. De plus, il a des qualités indéniables pour la manipulation des arbres et des listes. En effet, dans un langage procédural il faut programmer soi-même la gestion des listes (construction, suppression d'un élément,..;) en indiquant comment il faut procéder. Avec Prolog, il suffit de dire le QUOI et non le COMMENT.

Prolog effectue un raisonnement symbolique.

2.2. Les faiblesses du logiciel utilisé

L'interface du Prolog utilisé est suffisante mais plus élémentaire qu'avec, par exemple, Think Pascal ou Think C de Symantec.

Il est également assez lent.

3. La représentation des données

3.1. Comme arbre binaire

Une expression numérique a déjà été vue comme un arbre binaire au niveau conceptuel dans la seconde partie. C'est aussi comme arbre binaire que nous représenterons une expression. Par exemple, l'expression 4+5*(7-2) se représente par

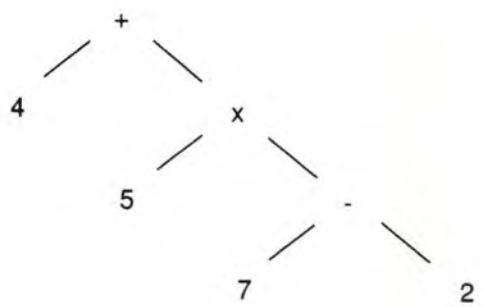


Figure 5.1

L' expression 4+5*(7-2) vue comme un arbre binaire

Cette représentation a l'avantage de pouvoir être manipulée aisément par Prolog. En effet, l'expression 4+5*(7-2), si elle est un terme composé, est d'office, pour lui, un arbre binaire.

3.2. Ce qu'il faut mémoriser

Nous mémoriserons

a les formes. Elles seront finalement stockées sur disque.
b des variables "globales"
- l'identifiant de l'utilisateur : contient toujours le nom de l'utilisateur en cours.
- l'expression étudiée qui est ainsi disponible en permanence.
- la valeur proposée par l'utilisateur pour une expression.
- la dernière expression fausse.
- un drapeau indiquant si une faute a été trouvée pour l'affichage des commentaires
- un drapeau indiquant si une confusion a été trouvée pour l'affichage des commentaires

c les valeurs des paramètres.
- type de recherche
- type de décomposition

d des listes
- La liste des expressions mal évaluées et, chaque fois, la valeur proposée par l'élève afin de rédiger les commentaires
- La liste de toutes les expressions étudiées par un élève pour contrôler les répétitions d'une même structure.

3.3. Persistance des données

Seules les *formes* et les valeurs des paramètres sont des données persistantes. Elles seront enregistrées dans le fichier de l'élève. Toutes les autres données sont temporaires et sont donc perdues en quittant le logiciel. N'oublions pas que les *familles* et les *noyaux* sont des notions dynamiques et sont dès lors recalculées chaque fois que l'on souhaite les voir apparaître.

4. Architecture générale

4.1. Configuration.

On peut faire un rapprochement avec un système expert. En effet, nous essayons dans le didacticiel de simuler une approche intelligente où un expert, le professeur, a introduit certaines connaissances dans la base de données. L'architecture générale peut, en effet, se voir comme suit :

Nous avons

- les deux fichiers qui, en fin de session d'exercices, seront modifiés en fonction de la base de faits.
- une base de faits composée en début de session des deux fichiers et qui s'enrichit des faits nouveaux construit lors de la session en cours.
- un module explication qui commente le déroulement de la session.
- un interface bien entendu.

4.2. Interactions.

L'interaction est représentée par une flèche qui est lue "fournit de l'information à"
Dans le schéma qui suit nous pouvons donc lire : Il y a un échange d'information dans les deux sens entre le fichier des élèves et la base de connaissances. Le fichier du professeur fournit de l'information à la base de connaissances sans en recevoir.

4.3. Schéma.

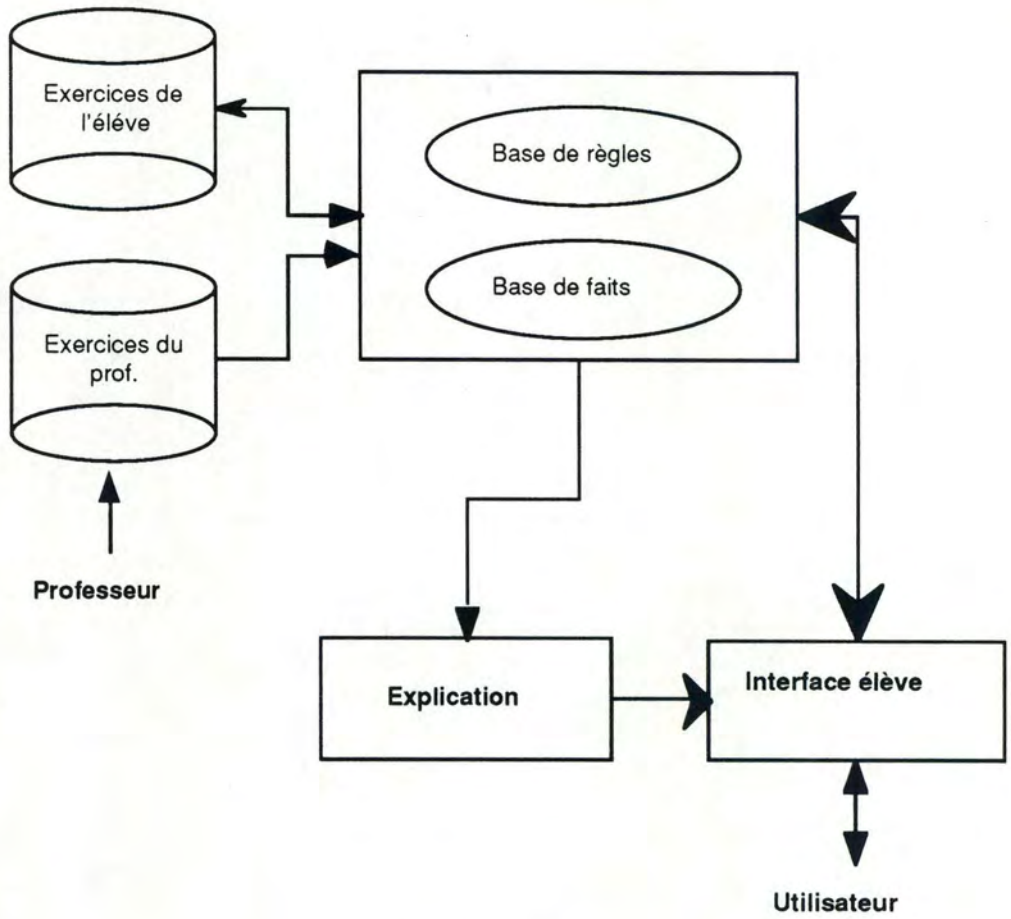


Figure 5.2

Le didacticiel vu comme un système expert

5. Structure de la documentation

5.1. Les modules

Le code est constitué d'un seul fichier. A un fichier sont associées plusieurs fenêtres. Chaque fenêtre pouvant être compilée seule peut être vue comme un module.

5.2. Le texte

Le texte du programme est présenté séparément (voir table des matières). Il est structuré suivant les modules. Au début de chaque module se trouve une table des matières relative à ce module.

5.3. Navigation à l'aide de la table des prédicats

En neuvième partie vous trouverez une table des prédicats qui est triée par ordre alphabétique avec référence de la page où il est décrit. Vous pouvez donc connaître le module auquel ce prédicat appartient. Pour ensuite le retrouver dans le code, vous allez trouver dans le code du module une table des matières de ce module et ainsi le situer aisément.

5.4. La documentation

5.4.1. Principe

Vous trouverez ci-dessous la liste exhaustive de tous les prédicats du programme. Il sont présentés par module. Ils sont presque tous commentés.

5.4.2. La documentation

La documentation est présentée comme ceci :

Module <Nom du module>

Prédicat(Arg1,arg2,arg3,...)

argument	**type**
Nom de l'argument	type de l'argument

Description
Description de l'action du prédicat

Remarque
Les remarques éventuelles

---

6. La documentation

1. Module Action des menus

Exercices(personnel)

argument	**type**
personnel	atom

Description
Ce prédicat est le but associé à l'item "personnel" du menu Exercice.
Il lance la création et l'analyse d'un exercice dont l'énoncé doit être rentré au clavier.

Exercices(à_travailler)

argument	**type**
à_travailler	atom

Description
Ce prédicat est le but associé à l'item "à travailler" du menu Exercice.
Il lance la création et l'analyse d'un exercice dont l'énoncé doit être choisi dans la liste qui est présentée.

Exercices(mes_bases)

argument	**type**
mes_bases	atom

Description
Ce prédicat est le but associé à l'item "mes bases" du menu Exercice.
Il lance la création et l'analyse d'un exercice dont l'énoncé doit être choisi dans la liste qui est présentée.

Exercices(du_professeur)

argument	**type**
du_professeur	atom

Description
Ce prédicat est le but associé à l'item "du professeur" du menu Exercice.
Il lance la création et l'analyse d'un exercice dont l'énoncé doit être choisi dans la liste qui est présentée.

Recherche(interactive)

argument	**type**
interactive	atom

Description
Ce prédicat est associé à l'item "interactive" du menu Recherche.
Il configure la recherche pour que l'ordinateur effectue la *recherche montante* .

Recherche(auto)

argument	**type**
auto	atom

Description
Ce prédicat est associé à l'item "auto" du menu Recherche.
Il configure la recherche pour qu'elle soit la *recherche directe* réalisée par l'ordinateur.

Divers(rapport_général)

argument	**type**
rapport général	atom

Description
Ce prédicat est le but associé à l'item "rapport général" du menu Divers.
Il lance la création d'un rapport général tel que décrit dans la partie deux. Il s'agit d'une consultation.

Divers(aide)

argument	**type**
aide	atom

Description
Ce prédicat est le but associé à l'item "aide" du menu Divers.
Il fait apparaître une fenêtre avec ascenseur qui permet de consulter les formules courantes.

Divers(Les_noyaux)

argument	**type**
rapport général	atom

Description
Ce prédicat est le but associé à l'item "les noyaux" du menu Divers.
Il fait apparaître une fenêtre avec ascenseur qui permet de consulter les noyaux.

Divers(configuration)

argument	**type**
configuration	atom

Description
Ce prédicat est le but associé à l'item "configuration" du menu Divers.
Il fait apparaître un dialogue prioritaire qui permet de changer des paramètres techniques.

Général(nouvel_utilisateur)

argument	**type**
nouvel utilisateur	atom

Description
Ce prédicat est le but associé à l'item "nouvel utilisateur" du menu Général.

Il fait apparaître un dialogue prioritaire qui permet de changer l'identifiant de l'utilisateur, de sauver le fichier de l'utilisateur sortant et de charger celui du nouvel utilisateur s'il existe déjà.

Général(quitter)

argument	**type**
quitter	atom

Description
Ce prédicat est le but associé à l'item "quitter" du menu Général.
Le fichier de l'utilisateur est sauvé et on quitte le didacticiel.

<ABOUT>(X)

Description
Ce prédicat est le but associé à l'item "About" du menu Pomme du Mac.
Il fait apparaître un dialogue prioritaire qui mentionne qu'il s'agit d'un mémoire avec une série de renseignements

remarque :
l'argument X est un argument purement technique sans signification.

---

mark_unmark(Arg1,Arg2)

argument	**type**	
arg1	atom	menu Recherche ou menu Décomposition
arg2	atom	référe l'item du menu de arg1

Description
Ce prédicat permet de cocher l'item choisi et de décocher celui qui l'était avant.

---

bselection(Arg1,Arg2)

argument	**type**	
arg1	variable	l'expresion à analyser
arg2	variable	le type d'exercices

Description
Ce prédicat qui est commun aux quatre types d'exercices gère la *décomposition manuelle.*
Elle nécessite des initialisations et des procédures propres à cette décompostion. Il prend en charge le premier appel d'un exercice avec ce mode de décompostion.

---

cselection

Description
Ce prédicat gère la décomposition manuelle à partir du deuxième exercice utilisant ce type de décomposition.

---

init_exercices(Dest)

argument	type	description
Dest	variable	indique l'origine de l'exercice.

Description
Ce prédicat effectue une série d'initialisations nécessaires avant d'entamer un nouvel exercice.

---

nonvide(arg1,arg2)

argument	type	description
arg1	liste	liste entrée.
arg2	liste	liste sortie

Description
Ce predicat est utilisé dans la construction des listes d'exercices. Il permet, lorsque la liste d'entrée est vide, de la remplacer par une liste d'un seul élément qui mentionne de choisir un exercice personnel ou du professeur.

2. Module Interface

Pour les prédicats suivants, le lecteur est renvoyé à la lecture du code où se trouvent les commentaires adéquats.

dialogueIdent(Btn,Rap)

dialogueexp(Btn, Rap)

essai(Choix, List, M)

suiteessai(Di, 3, Choix)

mondialogue(Btn, R, résultat, Val, X, Rep)

config(De1, De2, De3, De4, De5, R1, R2, R3, R4, Ve1, Ve2, Ve3, Ve4, Ve5, Vr1, Vr2, Vr3, Vr4)

mondialogue3(Btn, R, résultat, Text, Exp)

aide

creerhistorique

creerrapport

creerevolution

creertampon

creertampon2

effacerfenetres

effacertampon

effacertampon2

garnirhistorique

voirhistorique

effacerhistorique

garnirevolution(Z)

voirevolution

effacerevolution

monmessage(M)

3. Module Aide

Ce module ne contient aucun prédicat. Il s'agit d'une liste de formules.

4. Module Confusions

recherchefaute(X, Sortie)

argument	type	description
X	term	expression à évaluer
Sortie	term	expression qui est une fausse règle

Description
Ce prédicat reçoit l'expression X. Si la valeur proposée par l'élève est incorrecte, Sortie une expression dont l'évaluation donne la réponse incorrecte proposée par l'élève.

---

faute(Entree, Sortie, Valeur, M, A, B, C, D)

argument	type	description
Entrée	term	strucure de l'expression à évaluer
Sortie	term	fausse règle qui conduit au résultat de l'élève
Valeur	variable	valeur de Entrée
M	variable	numéro du type de structure considérée
A	variable	composante de la structure

B	variable	idem
C	variable	idem
D	variable	idem

Description
Ce prédicat reçoit Entrée, valeur et retourne les autres variables.

Reamarque
Si aucune fausse règle n'est trouvée, alors sortie est unifié avec "non"

---

entree(Entree, A, B, C, D, T, M)

argument	**type**	**description**
Entrée	term	structure de l'expression à évaluer
A	variable	composante de la structure
B	variable	idem
C	variable	idem
D	variable	idem
T	variable	type de la racine de l'expression à évaluer
M	entier	numéro d'ordre du type de l'expression

Description
Ce prédicat reçoit Entrée et retourne les autres variables où A,B,C,D sont les composantes de l'expression.
Recherche si l'expression dans le premier argument est une des formes prédéfinies et qui sont des formes à risques. Si oui, A,B,C,D,T et M sont unifiés.

Remarque
D est parfois unifié avec "rien"; Voir le code.

---

sortie(arg1, A, B, C, D,S)

argument	**type**	**description**
arg1	atom	type de l'expression à partir de laquelle on cherche une fausse règle
A	variable	composante de la structure
B	variable	idem
C	variable	idem
D	variable	idem
S	term	valeur proposée par l'élève

Description
Ce prédicat reçoit arg1 et retourne les autres variables.
Les cinq premiers arguments sont donnés. On cherche à unifier S qui représente lors une expression parfois utilisée quand il y a faute en lieu et place de l'expression correcte. Plusieurs possibilités existent pour S. C'est dans la procédure "faute" que la vérification de l'existence d'un bon S est faite.

Remarque
D est parfois unifié avec "rien". Voir code.

---

messageerreur(N, A, B, C, D, S)

argument	type	description
N	variable	numéro du type d'expression à analyser
A	variable	composante de la structure
B	variable	idem
C	variable	idem
D	variable	idem
S	term	valeur proposée par l'élève

Description
Ce sont les messages adéquats qui expliquent la démarche à suivre lorsque S a été trouvé.
Ces messages sont utilisés par le dialogue 4 dans sa seconde version pour afficher et ensuite demander s'il s'agit d'une confusion.

---

operationadd(A, B, A+B)

operationmul(A, B, A*B)

operationdivi(A, B, A/B)

Pour ces prédicats, le lecteur est invité à consulter le code.

---

listeconfusions(faute(E, S, V, N, A, B, C, D))

argument

faute(E, S, V, N, A, B, C, D) voir prédicat faute.

Description
Ce prédicat permet de mettre à jour la liste des expressions à évaluer en y ajoutant E et la liste des expressions relevant d'une fausse règle en y ajoutant S. Ces deux listes sont tenues à jour en parallèle.

---

voirconfusions

Description
Ce prédicat fait lire chaque élément de la liste des confusions.

---

lireconfusions (X)

argument	type
X	variable

Description
Ce prédicat lit une confusion et l'affiche dans la fenêtre historique.

init_exercices(Dest)

argument	type	description
Dest	variable	origine de l'exercice

Description
Ce prédicat réalise une série d'initialisations communes aux quatre types d'exercices. Son exécution est obligatoire.

---

creer_exercice_trav(X)

argument	type	description
X	term	contient l'expression à analyser

Description
Ce prédicat réalise une série d'opérations propres à la création d'un exercice extrait de la liste des exercices dont la connaissance est petite. Il s'agit de sélectionner les structures dont la connaissance est faible, d'afficher cette liste, puis d'en extraire celle choisie par l'élève.

---

creer_exercice_prof(X)

argument	type	description
X	term	contient l'expression à analyser

Description
Ce prédicat réalise une série d'opérations propres à la création d'un exercice extrait de la liste des exercices du professeur. Il s'agit de sélectionner les structures dont la connaissance est faible, d'afficher cette liste puis d'en extraire celle choisie par l'élève.

---

creer_exercice_pers(X)

argument	type	description
X	term	contient l'expression à analyser

Description
Ce prédicat réalise une série d'opérations propres à la création d'un exercice dont l'énoncé est rentré au clavier.

---

creer_exercice_mes_bases(X)

argument	type	description
X	term	contient l'expression à analyser

Description
Ce prédicat réalise une série d'opérations propres à la création d'un exercice extrait de la liste des exercices de base. Il s'agit de sélectionner les *familles*, d'afficher cette liste puis d'en extraire celle choisie par l'élève.

---

simp(E, F)

argument	type	description
E	term	contient l'expression à analyser
F	variable	la structure de l'expression E

Description
Ce prédicat dépouille un arbre de ses nombres pour conserver uniquement la structure.
Il reçoit E et retourne F.

Exemple
Si E=4+6*(1-5) alors on a F=_+_*(_+_)

---

garnir(E, F)

argument	type	description
E	term	contient la structure de l'énoncé
F	term	la structure garnie de nombres

Description
Ce prédicat garnit une structure avec des nombres aléatoires non nuls.

Exemple
Si E=_+_*(_+_), alors F=4+9*(5-3) par exemple.

---

sgarnir(E, F)

argument	type	description
E	terme	contient la structure de l'énoncé
F	terme	la structure garnie de lettres

Description
Ce prédicat garnit une structure avec des lettres.

Remarque importante
La structure ne peut avoir plus de deux niveaux en plus de la racine.

---

ssimp(E, F, N)

argument	type	description
E	terme	contient la structure de l'énoncé
N	entier	nombre de niveaux conservés en plus de la racine
F	terme	une structure

Description
Ce prédicat tronque une structure en ne conservant que les N premiers niveaux.

Exemple
Si E=4+3*(1+1/2) et N=2, alors F=_+_*_.

---

listeformes(X)

argument	type	description
X	Variable	une liste

Description
Ce prédicat renvoie la liste des familles

---

listeformes(X,Y)

argument	type	description
X	liste	une liste de structures
Y	liste	liste de structures tronquées et garnies de lettres

Description
Ce prédicat reçoit le liste des *formes* dans X et renvoie la liste des *familles* dans Y.

---

decapiter(E, F)

argument	type	description
E	term	une expression numérique
F	variable	une structure

Description
Ce prédicat tronque une expresion numérique en ne conservant que les deux premiers niveaux en plus de la racine. C'est une forme simplifiée de ssimp.

---

garnirlettre(E, F)

argument	type	description
E	term	une structure
F	variable	structure garnie de lettres

Description
Ce prédicat crée une liste de lettres puis appelle sgarnir qui va utiliser la liste qui vient d'être créée.

mettreforme(L, F)

argument	type	description
L	Variable	un couple (expression, connaissance)
F	variable	expression précédée de la connaissance

Description
Ce prédicat formatte les éléments de la liste qui sera affichée à l'écran et qui sera la liste des expressions dont la connaissance est petite précédée de leur connaissance.

defaire(A, B)

argument	type	description
A	variable	élément choisi par l'élève dans la liste présentée à l'écran
B	variable	élément de la liste dépouillé de son préfixe

Description
Ce prédicat enlève le préfixe qui avait été placé par mettreforme.

keepAfter(Arg1, Arg2, Arg3)

argument	type	description
Arg1	Liste	liste dans laquelle on cherche Arg2
Arg2	Liste	caractère recherché
Arg3	Variable	ce qui suit le caractère recherché

Description
Ce prédicat enlève le premier caractère d'une liste jusqu'au moment où le caractère suivant est celui contenu dans le second argument qui est donné.

selectionfaiblesses(Critere, A)

argument	type	description
Ctritere	numérique	valeur plafond acceptée pour la connaissance
A	couple	couple (connaissance,structure garnie d'entiers)

Description
Ce prédicat sélectionne une forme dont la connaissance est petite.

6. Module Paramètres

Ce module ne contient aucun prédicat. Il renferme les valeurs des paramètres sous forme de faits.

enregistrer2(Form, Exp, Orig)

argument	type	description
Form	terme	expression numérique
Exp	entier	succès (1) ou échec(0)
Orig	atom	origine de la structure

Description
Ce prédicat enregistre une *forme*.
Il distingue les cas où l'argument structure de "forme" n'existe pas encore et celui où aucun prédicat "forme" ne possède cette srtucture comme argument.

---

majcpt(AcptB, AcptF, NcptB, NcptF, Exp)

voir le code

---

majprovenance(Aprov, Nprov, pers)

voir le code.

---

enregistrerfaux

Description
Ce prédicat enregistre la structure *où naît l'erreur*.
Un message est affiché dans la fenêtre Historique.

---

effacerfauxvrai

Description
efface les variables globales contenant la dernière expression bonne et la dernière expression fausse.

---

miseajour(Id, Var)

argument	type	description
Id	atom	nom de la variable globale.
Var	atom	valeur de la variable globale.

Description
Ce prédicat met à jour la variable globale Id en lui donnant la valeur Var.

---

miseajourp(Id(Var))

argument	type
Id(Var)	prédicat

Description
Ce prédicat met à jour le fait Id(Var) où Id est le prédicat et Var son argument.

---

8. Module gestion des fichiers

charger(X)

argument	type	description
X	atom	nom de l'utilisateur

Description
Ce prédicat charge le fichier de nom X. Si aucun fichier n'est trouvé, ce prédicat ne fait rien et réussit.

---

chargerprof(D,R)

argument	type	
D	liste	liste de départ
R	liste	liste des exercices du prof non encore faits

Description
Ce prédicat réussit toujours. R contient la liste de tous les exercices du professeur dont la structure n'est pas encore présente dans la base de faits. D est la liste de départ qui sera augmentée de R.

---

stockerdisk(X)

argument	type	description
X	atom	nom de l'utilisateur

Description
Ce prédicat stocke sur disque tous les faits du type forme en créant un fichier du nom X. L'ancien fichier de même nom , s'il existe, est écrasé.

---

inscrire(X)

argument	type
X	variable

Description
Ce prédicat inscrit un fait X à la fin du fichier de l'utilisateur courant.

sauver_source(X)

argument	type
X	atom

Description
Ce prédicat sauve tout le fichier source du logiciel sous le nom copie de secours suite à un changement des paramètres via l'item "configuration" du menu "Divers".

9. Module Rapport

rapportg

Description
Ce prédicat lance la réalisation du rapport général concernant un élève.

ecrirefamille(X)

argument	type	description
X	variable	une famille

Description
Ce prédicat écrit dans la fenêtre du rapport une ligne qui mentionne une famille où l'élève rencontre des difficultés. Ce prédicat est donc appelé chaque fois qu'une ligne du rapport concernant une famille est à écrire.

attacheverdict(X)

argument	type	description
X	variable	couple (jugement,forme)

Description
Ce prédicat attache un jugement à une structure garnie pour pouvoir ultérieurement l'afficher.

ecrireverdict(X)

argument	type	description
X	variable	couple (jugement,forme)

Description
Ce prédicat écrit une ligne constituée d'une structure garnie suivie du jugement qui la concerne dans la fenêtre du rapport.

verdict(Con, L, Cb, Cf, Message)

argument	type	description
Con	variable	la connaissance
L	liste	les dix derniers résultats
Cb	entier	nbre de fois que la structure a été rencontrée et bien évaluée
Cf	entier	nbre de fois que la structure a été rencontrée et mal évaluée
Message	atom	un message

Description
Ce prédicat attache effectivement un jugement à la structure concernée ayant la connaissance C.
Les 3 premiers prédicats sont des données. Le 4ième est unifié avec le message qui est le verdict.

La première clause concerne le cas où la connaissance est supérieure au seuil en paramètre. Implicitement ce seuil ne sait être atteint que si le nombre d'expériences est suffisant ! Si elle est vérifiée, la procédure est arrêtée. Sinon,

La seconde clause :
Cas où la connaissance n'est pas "bonne".
Elle teste si le nombre d'expériences est suffisant. Si ce n'est pas le cas, la procédure s'arrête avec le message adéquat. Sinon,

La troisième clause :
Cas où la connaissance est insuffisante et où le nombre d'expériences est trop petit.
Elle compare la connaissance des dernières expériences avec la connaissance générale. Si elle est supérieure d'un demi point à la connaissance générale on déclare qu'il y a progrès. Sinon,

La quatrième clause :
Cas où la connaissance est insuffisante et où le nombre d'expériences est trop petit et où il n'y a pas progrès.
Idem que plus haut, mais si la connaissance sur les dernières expériences est inférieure d'un demi point à la connaissance générale, on déclare qu'il y a recul. Sinon,

La cinquième clause :
Il reste la dernière possibilité : évolution incertaine.

Module Affichage des commentaires

erreur

Description
Ce prédicat lance la recherche des commentaires à afficher dans le cadre d'une *recherche automatique.*

---

fauteindet(X)

argument	type	description
X	term	expression où nait l'erreur

Description
Ce prédicat lance une petite analyse de l'erreur si on n'a pas trouvé de fausse règle.

---

analyse_erreur(X)

argument	type	description
X	term	expression où nait l'erreur

Description
Ce prédicat s'occupe d'afficher les messages qui concernent uniquement les erreurs de calculs dans le cas où aucune fausse règle n'a été détectée.

---

afficherformule(X)

argument	type	description
X	term	expression où nait l'erreur

Description
Ce prédicat afiche la formule qui a été mal appliquée uniquement dans le cas où analyse_erreur n'a rien fait.

---

maide(arg1,arg2)

argument	type	description
arg1	atom	formule à développer
arg2	atom	un message

Description
Ce prédicat permet d'obtenir dans arg2 un message adéquat qui indique la bonne façon de développer l'expression donnée dans arg1.

---

listerreur(X)

argument	type	description
X	term	expression mal évaluée

Description
Ce prédicat enregistre une expression mal évaluée ou non évaluée X en tête d'une liste appelée "expfausse". Il crée la liste si elle n'existe pas encore.
Ce prédicat n'est utilisé que dans le cadre de la d*écomposition manuelle*

---

listevalfausse(X)

argument	type	description
X	term	valeur erronée

Description
Ce prédicat enregistre une valeur proposée par un élève et qui est incorrecte. Elle est placée en tête d'une liste qui s'appelle "valfause". Cette liste est créée si elle n'existe pas.
Ce prédicat n'est utilisé que dans le cadre de la d*écomposition manuelle*

---

voirerreur

Description
Ce prédicat appelle les deux listes "expfausse" et "valfausse", les renverse pour que les éléments soient dans l'ordre réel où ils ont été rencontrés, puis affiche dans la fenêtre "Historique" ces deux listes en parallèle.
Ce prédicat n'est utilisé que dans le cadre de la d*écomposition manuelle*

---

lerreur(X)

argument	type	description
X	term	expression mal évaluée

Description
Ce prédicat écrit une ligne composée de l'expression mal évaluée et de la valeur proposée par l'élève ou d'un message disant Que l'élève a renoncé à l'évaluer. Ce prédicat est utilisé par *voirerreur.*
Ce prédicat n'est utilisé que dans le cadre de la d*écomposition manuelle*

---

trouverlg(X)

argument	type	description
X	term	expression mal évaluée

Description
Ce prédicat calcul la longueur de X et stocke sa valeur dans une variable globale qui garde ainsi en mémoire la plus grande longueur des expressions rencontrées. Ceci permet un alignement correcte de l'affichage avec des caractères non proportionnels.
Ce prédicat n'est utilisé que dans le cadre de la d*écomposition manuelle*

Module Extraction des sous expressions

explorer(X)

argument	type	description
X	term	expression à analyser

Description
Cette procédure explore l'expression vue comme un arbre binaire.
Cette procédure explore l'arbre de l'expression à évaluer et s'arrête dès que la sous expression *où naît l'erreur* est découverte ou que l'exploration est terminée.
La procédure comporte 11 clauses.

La première teste si on n'a pas affaire à un entier ou une fraction d'entiers. Dans ce cas, on a terminé. Sinon, on poursuit en interrogeant l'utilisateur sur la valeur de l'expression en cours. Si l'expression est bien évaluée, c'est terminé, sinon on passe à la clause suivante.

La seconde clause met à jour la variable vrai si c'était un entier ou une fraction d'entiers.

La troisième clause teste si la racine de l'arbre est une addition. Si l'expression a mal été évaluée, on passe au sous arbre gauche. On force ensuite à l'exploration du sous arbre droit. On passe enfin à la clause suivante.

La quatrième clause explore le sous arbre droit uniquement si l'arbre est faux et son sous arbre gauche vrai. En effet, on n'a pas trouvé d'erreur à gauche, donc elle se situe peut-être à droite.

Les clauses suivantes, sauf la dernière, font le même travail pour les autres racines.

La onzième clause est toujours vraie. Il est important de remarquer que l'on atteint cette clause si on a affaire à un entier ou une fraction d'entiers. Ceci permet, en fin de parcours, de forcer la réussite de manière à ne pas rendre la main au dialogue.

---

dialoguereponse(X, E, Rep)

argument	**type**	**description**
X	term	expression à analyser
E	entier(fract)	valeur de X
Rep	variable	

Description
Ce prédicat permet de choisir le type de dialogue avec l'élève suivant la valeur du paramètre "reponseentree". Ou la réponse est saisie, ou on la propose et l'élève répond oui s'il a cette valeur, non dans l'autre cas.

---

mouinon(X,E,Rep)

argument	**type**	**description**
X	term	expression a évaluer
E	term	valeur de X
Rep	variable	

Description
Ce prédicat permet de savoir si l'élève a ou n'a pas la valeur E pour l'expression X. Si ce n'est pas le cas, Rep est unifié avec "non". La variable globale "vrai" ou "faux" est mise à jour suivant que la réponse est correcte ou non.

---

verificationresultat(D, X, E, Rep)

argument	type	description
D		référence le dialogue
X		expression à analyser
E		valeur de X
Rep		

Description
Ce prédicat permet de savoir si l'élève a ou n'a pas la valeur E pour l'expresion X. Si ce n'est pas le cas, Rep est unifié avec "non". La variable globale "vrai" ou "faux" est mise à jour suivant que la réponse est correcte ou non.

---

auto(E)

argument	type	description
E	term	expression à analyser

Description
Ce prédicat lance la localisation directe d'une sous expression dont la probabilité est élevée d'être la source de l'erreur.

---

listessexp(E,L)

argument	type	description
E	term	expression à analysée
L	liste	liste des structures ds la base de données

Description
Ce prédicat permet la construction de la liste des structures déjà vues qui sont sous expression de l'expression donnée et qui satisfont à des critères de sélection portant sur la connaissance et le nombre de fois que ces structures ont été rencontrées.

---

lss(X,Es)

argument	type	description
X	term	structure
Es	term	expression numérique

Description
Ce prédicat reçoit Es. Il cherche ensuite une structure qui est déjà vue et qui est contenue dans X avec une connaissance "petite" (voir code) et rencontrée suffisamment de fois.

---

si(E,Sexp)

argument	type	description
E	term	.structure
Sexp	term	structure

Description
Ce prédicat vérifie si une structure E contient un sous structure Sexp.

---

suiterep(Rep,E,L)

argument	type	description
Rep		
E	term	expresion à analyser
L	liste	liste de ttes les ss expr de E susceptibles d'être la source de l'erreur

Description
Ce prédicat va proposer à l'élève le premier élément de la liste comme source de l'erreur.

---

analyserep(E, ValE, Rep)

argument	type	description
E	term	expression à analyser
ValE	entier(frac)	valeur de E
Rep		drapeau technique

Description
Ce prédicat permet de recueillir la réponse de l'élève à la proposition "avez-vous E=ValE ?".

---

bonneselection2(E, Se)

argument	type	description
E	term	expression ds laquelle on sélectionne
Se		expresion sélectionnée

Description
Ce prédicat permet de remplacer une expression dont une partie a été sélectionnée et qui est syntaxiquement valide par une nouvelle expression où la partie sélectionnée a été remplacée par sa valeur.
Ce prédicat n'est utilisé que dans la *décomposition manuelle*.

---

divers(Dial, N, R, Text, Exp)

argument	type	description
Dial		référence le dialogue
N		numéro du bouton
R		réponse proposée
Text		la sélection
Exp		valeur de la sélection

Description
Procédure appelée par le dialogue 6.
Lance les procédures liées au dialogue de saisie d'une réponse quand la décomposition se fait par sélection. (Voir aussi le code). Il y a le contrôle syntaxique et ensuite des actions différentes suivant le bouton sur lequel on clique. On peut laisser l'ordinateur calculer puis remplacer la sélection. On peut proposer une valeur pour la sélection qui, si elle est bonne, provoque le remplacement de l'expression contenant la selection par l'expression réduite; Enfin on peut quitter l'exercice.
Ce prédicat n'est utilisé que dans la *décomposition manuelle.*

Module Utilitaires

fraction(X,Y)

argument	type	description
X	term	expression fractionnaire
Y	liste	liste formée du numérateur et du dénominateur

Description
Ce prédicat transforme un nombre fractionnaire en une liste de deux éléments : [numérateur,dénominateur] et un nombre en une liste [nombre,1]. Ainsi on ne manipule que des fractions.

Les prédicats suivants manipulent des fractions entre elles en conservant le résultat sous forme fractionnaire à l'aide de listes de deux éléments : [numérateur,dénominateur].

plus(A+B, G)

moins(A-B, G)

fois(A*B, G)

divi(A/B, G)

---

eval(X,R)

argument	type	description
X	term	expression a évaluer
R	liste	[numérateur, dénominateur]

Description
Ce prédicat évalue la valeur d'une expression sous forme d'une liste de deux éléments qui représentent un nombre fractionnaire.

---

pgcd(I,J,K)

argument	type	description
I	entier	
J	entier	
K	entier	K est le pgcd de I et J

Description
Ce prédicat calcule le pgcd de deux nombres à l'aide de l'algorithme d'Euclide.
La première clause exprime que le pgcd de deux nombres dont un est nul est l'autre nombre, ce qui est évident
La seconde clause réduit le problème en exprimant que le pgcd de deux nombres est le même que le pgcd du reste de la division de l'un des deux par l'autre et l'autre. Il s'agit d'une réduction car on tombe fatalement sur le cas ou le second argument du prédicat est nul.
Par exemple, pgcd(12,4)= pgcd(4,0)=4
pgcd(12,5)= pgcd(5,2)= pgcd(2,1)= pgcd(1,0)=1
pgcd(5,12)= pgcd(12,5)=...

---

reduire(X,R)

argument	type	description
X	liste	les deux éléments représentant la fraction à réduire
R	liste	les deux éléments représentant la fraction réduite

Description
Ce prédicat calcule une liste R qui représente la fraction réduite représentée par la liste X.

---

connaissance(OldC,Cl,Nc,[Exp| Rest],Exp)

argument	type	description
OldC	réel	valeur de la connaissance avant nvelle évaluation
Cl	liste	liste des dix derniers résultats avant nvelle évaluation
Nc	réel	valeur de la nouvelle connaissance
[Exp| Rest]	liste	nouvelle liste des dix derniers résultats
Exp	entier	nouveau résultat d'une estimation

Description
Ce prédicat calcule la valeur de la nouvelle connaissance Nc à partir de OldC,Cl et Exp.

---

remove_last(L1,L2)

argument	type	description
L1	liste	liste avant évaluation
L2	liste	liste L1 dont le dernier élément a été enlevé.

Description
Ce prédicat transforme une liste en l'amputant de son dernier élément.

---

connaissancelp(L, C)

argument	type	description
L	liste	liste des dix derniers résultats
C	liste	connaissance sur une liste partielle issue de L.

Description
Ce prédicat recherche la connaisance calculée sur une liste partielle issue de L.

---

listepartiel(L, Lg, N)

argument	type	description
L	liste	liste des dix derniers résultats
Lg	entier	longueur de la liste partielle à extraire.
N	liste	liste partielle issue de L

Description
Ce prédicat extrait les N premiers éléments de L.

---

lp(L, D, Cptdepart, Lg, S)

argument	type	description
L	liste	liste des dix derniers résultats
D	liste	liste de départ.
Cptdepart	liste	compteur
Lg	entier	valeur max du compteur
S	liste	liste extraite

Description
Ce prédicat extrait les N premiers éléments de L et l'ajoute à la fin de D pour donner S.

---

sousarbre(E, Sexp, Mod)

argument	type	description
E	term	expression numérique
Sexp	term	expression numérique extraite de E
Mod	term	structure recherchée dans E

Description
Ce prédicat recherche si une expression E ne contient pas une structure Mod et si c'est le cas, donne dans Sexp la sous expression trouvée.

---

configuration

Description
Ce prédicat est le but associé à l'item "configuration" du menu "Divers". Il permet de changer la valeur de certains paramètres.

Les prédicats suivants sont directement liés au prédicats configuration. Je propose au lecteur de consulter le code.

valeurB1(B1, R1, R2)

valeurB2(B2, R3, R4)

validationg(D, B, Ve1, Ve2, Ve5)

validation(Ve1)

validation2(Ve2)

validation3(Ve5)

Les prédicats qui suivent sont facilement compréhensibles à la lecture du code. Je renvoie donc le lecteur à cette lecture.

compilation

lesmiseajours(Ve1, Ve2, Ve3, Ve4, Ve5, Vr1, Vr2, Vr3, Vr4)

random(N)

afficher(X, E)

ecrire(R, E)

identifier

---

test(X, D, D)

argument	**type**	**description**
X	term	expression numérique du fichier du prof
D	term	expression numérique extraite de E
D	term	structure recherchée dans E

Description
Ce prédicat est appelé lors du chargement du fichier du professeur via "selection".

---

testenonce(Form)

argument	**type**	**description**
Form	term	expression numérique

Description
Ce prédicat teste si une expression posséde une structure déjà connue ou non. Ce test n'a lieu que si le paramètre "fairetest2" le permet.

---

fractionentiere(X)

fractionentiere2(X)

argument	type	description
X	term	expression numérique

Description
Ces prédicats testent si une expression est une fraction entière ou non. Le second prédicat fait apparaître un message signalant qu'une fraction entière n'est pas admise. Ce second prédicat est utilisé pour éviter de sélectionner une fraction entière dans la *décomposition manuelle*.

---

souvenir

Description
Ce prédicat réussit toujours. Il permet de mémoriser dans une variable globale la liste des structures déjà vues au cours d'une session d'exercices d'un élève. Quand un élève se propose de lancer un nouvel exercice, ce prédicat recherche dans la liste des structures déjà vues si cette structure est présente. Si c'est le cas, la connaissance de cette structure est testée pour voir si elle est bonne. Si oui, on affiche un message destiné à l'élève pour lui dire qu'il connaît déjà bien cette structure et qu'il est peut-être inutile de poursuivre l'exercice.

---

pertinence(X, Lg)

argument	type	description
X	term	expression numérique
Lg	entier	le nombre de fois que X a déjà été rencontré

Description
Ce prédicat teste si une expression posséde une structure déjà connue ou non. Ce test n'a lieu que si le paramètre "fairetest2" le permet.

---

solution(X)

argument	type	description
X	term	expression numérique

Description
Ce prédicat lance la recherche de la localisation de l'erreur avec la décomposition automatique ou *recherche montante*.

---

killdouble(L, M, D)

killdouble(L, M)

argument	type	description
L	liste	liste des structures de la base
M	liste	L moins D
D	liste	liste des éléments présents au moins deux fois

Description
Ces prédicats permettent de dresser la liste des familles

---

double(Arg1, Arg2, Arg3, Arg4, Arg5)

double(Arg1, Arg2, Arg3)

argument	type	description
Arg1	liste	liste d'entrée
Arg2	liste	liste servant à accumuler
Arg3	liste	liste des éléments qui ne sont présents qu'une fois
Arg4	liste	liste servant à accumuler
Arg5	liste	liste des éléments présents au moins deux fois

Description :
Ce prédicat dresse la liste des éléments d'une liste donnée en entrée qui sont présents au moins deux fois et ceux qui ne sont présents qu'une fois.

---

killtriple(D, R, T)

Consulter le code.

---

noyau(T)

argument	type	description
T	liste	les noyaux

Description
Ce prédicat lance la construction de la liste des noyaux. Ceux-ci ne sont donc pas mémorisés, mais reconstruits chaque fois.

---

leNoyau(L)

argument	type	description
L	liste	les noyaux ou un message

Description
Ce prédicat lance la construction de la liste des noyaux. Si aucun noyau n'est trouvé, L est unifié avec un message signalant l'absence de noyau.

---

voir(List, M)

Affiche la liste dans un dialogue. Vous pouvez consulter le code pour le détail.

---

voirnoyau

Description
Ce prédicat lance la recherche des noyaux, puis affiche la liste.
Considérons la liste A=[1,2,3,2,6,5,2,3,1,4,4,2,6,2].
Fixons-nous comme but de trouver les éléments de A qui sont au moins en triple.
Soit Killdouble(A,B,C) qui est tel que :
A est la liste donnée
B est la liste sans double. Ici B=[1,2,3,6,5,4].
C est l'ensemble C=A\B=[2,2,3,1,4,2,6,2] soit l'ensemble des éléments présent au moins deux fois. Les triples, s'ils existent, sont dans cette liste.
On réapplique killdouble à C. Alors le troisième argument sera l'ensemble des éléments représentés au moins trois fois. On applique encore killdouble à ce dernier ensemble et le second argument est la liste des éléments, sans double, représentés au moins trois fois et que nous appelons le noyau.
C'est cette voie que nous avons utilisée pour déterminer les structures à retravailler pour un élève. Alors A est la liste des structures faibles de l'élève.

Les prédicats qui suivent sont de petits utilitaires très simples. Je vous invite à consulter le code pour leurs significations.

modeevolution(X, E)

selection(D, X, S)

solutiong(X)

validationgg(D, B, Ve1, Ve2, Ve5)

transfo(A, S)

membre(X, [Y| __1])

Module Syntaxe

Commentaires
On sait que Prolog est un langage très bien adapté à tous les problèmes d'analyse syntaxique de langages naturels ou formels. Une notation spécifique à cette analyse a été développée : DCG (Definite Clause Grammars). Cette notation est ensuite convertie en clauses ordinaires de Prolog. Le formalisme DCG se situe donc au dessus de Prolog. C'est ce langage qui a été employé pour réaliser la vérification syntaxique d'une expression. Une règle DCG ressemble à une clause normale de Prolog, mais a la forme particulière suivante :

tête --> corps.

syntaxe2(D, Btn, S)

syntaxe3(D, Btn, S)

argument	type	description
D	entier	référence du dialogue
Btn	entier	numéro de bouton
S	liste	liste de code ASCII

Description
Le prédicat "syntaxe2" est appelé par le dialogue numéro 2. Le prédicat "syntaxe3" est utilisé lors de la sélection d'une expression. Tous deux lancent la vérification de la syntaxe de la liste S.S'il y a une erreur un message apparaît dans le dialogue pour le signaler. S'il n'y a pas d'erreur une seconde vérification est faite pour contrôler que l'expression introduite au clavier pour "syntaxe2" ou l'expression sélectionnée pour "syntaxe3" n'est pas un entier ou un nombre fractionnaire.

---

syntaxerep2(D, N, Entree, X, V, Rep)

argument	type	description
D	entier	référence du dialogue
N	entier	numéro de bouton
Entree	term	réponse entrée par l'élève
X	term	expression à évaluer
V	liste	valeur de l'expression X
Rep	atom	drapeau

Description
Ce prédicat est appelé par le dialogue numéro 4 . Il permet des actions différentes suivant la "version" dans laquelle se trouve le dialogue ainsi que le bouton sur lequel l'élève a cliqué. Il est utilisé d'une part pour saisir la réponse de l'élève, puis dans une seconde version pour confirmer ou non l'usage d'une fausse règle. Consulter le code pour plus de détails.
Pour les actions déclenchées par les boutons, on a :

Bouton OK
Il vérifie la syntaxe
Si elle est correcte
Il met à jour la variable globale contenant la réponse de l'élève
Il vérifie si la réponse est correcte
Si oui le dialogue est fermé
si non il recherche si une confusion a eu lieu
si une confusion est trouvée
Les labels des boutons sont changés (oui/non)
une fausse règle est proposée
Si la syntaxe est incorrecte
un message est affiché
le dialogue reste ouvert en attente d'une nouvelle réponse.

Bouton Blanc
Il permet de quitter l'exercice par abandon

Les prédicats qui suivent sont commentés dans le code. Certains sont très courts et compréhensibles à leur lecture.

tst_phrase(Error, [Message, Prefix, Pos, Queue], G, I)

tst_syntaxe(Message, Queue, Queue)

exp(E)

add_exp(E)

add_exp_q(D, E)

add_exp_q(D, D, X, X)

mul_exp(E)

mul_exp_q(D, E)

mul_exp_q(D, D, X, X)

int_exp(E)

digits(S)

tdigits([S| T])

ndigits([S| T])

ndigits([], X, X)

digit(C)

add_op(+)

mul_op(*)

l_par

r_par

w

w(X, X)

testexpresion(I)

expreponse(E)

add_exp2(E)

mul_exp2(E)

mul_exp_q2(D, E)

```
mul_exp_q2(D, D, X, X)
int_exp2(I)
digits2(S)
tdigits2([S| T])
ndigits2([S| T])
ndigits2([], X, X)
digit2(C)
mul_op2(/)
```

Sixième partie

Guide pratique de l'élève

---

1. Avertissement

Il est nécessaire d'être un peu familiarisé avec le système d'exploitation du Macintosh.

Le logiciel utilisé pour réaliser ce didacticiel n'est pas une version professionnelle et ne possède donc pas toujours les qualités voulues d'un logiciel commercial. C'est ainsi que par moments, l'exécution vous semblera lente. C'est notamment le cas lorsque des dialogues doivent apparaître à l'écran.

2. Démarrage

Double-cliquons sur "AEEN.code". Après un temps d'attente, un dialogue vous demande de vous identifier. Ce nom sera celui de votre fichier qui stockera des informations vous concernant. Pour être efficace et cohérent, il faudra toujours utiliser ce nom au cours de toutes vos sessions d'exercices.

3. Déroulement d'un exercice.

Les premiers choix

Faire un exercice suppose dès le départ trois choix :

- sur la décomposition
- sur le type de recherche
- la façon dont l'exercice est choisi

Ces choix se font via les menus. Il y a deux sortes de menus : ceux qui déclenchent une action et ceux qui déterminent des choix sans entraîner d'action.

Vous devez d'abord décider si l'ordinateur va essayer de trouver directement, sans votre aide, l'endroit où vous auriez fait une erreur ou si l'ordinateur va essayer de situer, petit à petit, à l'aide de vos réponses,où se trouve votre erreur. ,

1. La recherche.

Vous choisissez entre un essai de localisation directe de votre erreur ou une recherche plus lente, mais plus sûre, en vous interrogeant.

On change de choix en déroulant le menu Recherche.

2. La décomposition.

Vous devez choisir :

- soit vous laissez le soin à l'ordinateur de découper votre énoncé en expression plus petite pour trouver le lieu où l'erreur s'est produite
- soit c'est vous qui allez découper votre énoncé en en extrayant vous-même une expression plus petite à l'aide d'une sélection.

3. Le type d'exercice.

Un exercice est

- soit rédigé par vous-même
- soit choisi parmi vos faiblesses qui sont connues par l'ordinateur.
  Il faut donc que ce ne soit pas la première fois que vous utilisiez le logiciel, car alors la liste est vide.
- soit choisi parmi les exercices proposés par le professeur. Ici, seuls apparaissent les exercices que vous n'avez pas encore faits.
- soit choisi parmi les expressions littérales proposées et qui sont basées sur vos faiblesses

On fait son choix en déroulant le menu Exercices.

Faire un exercice.

Exercice "personnel".

Un dialogue vous invite à rentrer une expression. Faites-le et ensuite cliquez sur OK.

La syntaxe est incorrecte

Si la syntaxe est incorrecte,un message apparaît signalant la présence d'une erreur et un signe # est inséré devant le premier caractère incorrect. Il suffit de modifier l'expression et de recliquer sur OK.

L'écran.

Deux fenêtres apparaissent alors : Historique et Évolution ainsi qu'un dialogue pour saisir votre réponse. Les fenêtres resteront à l'écran tant que l'on ne change pas d'utilisateur, alors que le dialogue s'efface et puis réapparaît à chaque étape de l'exercice.

Le dialogue

Il dépend du choix adopté pour la décomposition de l'expression.

1 L'ordinateur décompose.

- un dialogue oui-non La réponse est proposée à l'utilisateur.
- un dialogue pour saisir la réponse Si une erreur vous est proposée, il vous permet de signaler si elle a bien été commise ou non.

2. C'est vous qui décomposez.

Dans ce cas, un dialogue va apparaître pour gérer les sélections que vous allez faire.
Il propose trois choix :
- introduire une valeur. La syntaxe sera alors vérifiée.
- quitter
- donner sa langue au chat. La bonne valeur de la sélection est automatiquement mise à la place de la sélection.

Exercice "à travailler"

Ici, l'exercice n'est pas introduit au clavier mais choisi parmi une liste affichée à l'écran. Chaque exercice est préfixé par un nombre plus petit que 1 attaché à cet exercice. Ce nombre représente votre degré de connaissance de ce type d'énoncé. Plus il est proche de 1 et plus vous êtes bon pour ce type d"énoncé. D'ailleurs si vous êtes suffisamment bon, ce genre d'exercices n'apparaitra plus dans la liste. Le déroulement est ensuite le même que pour le point précédent.

Exercice "mes bases"

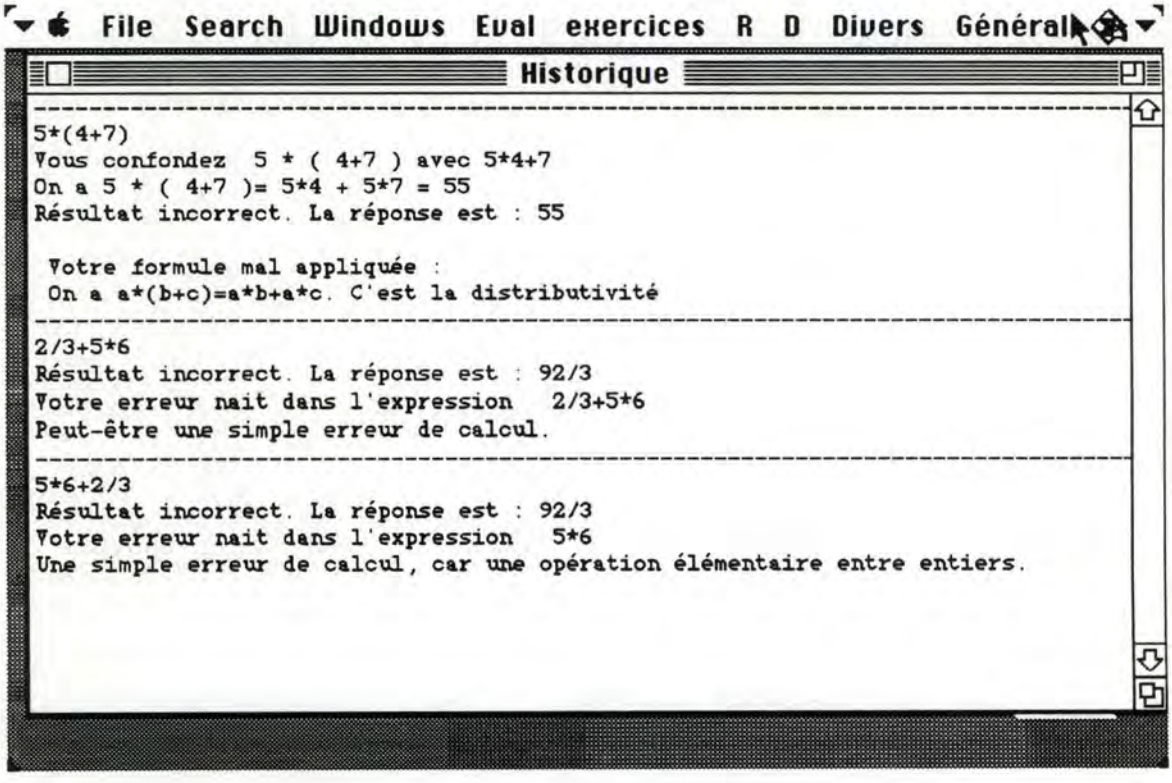
Ici aussi une liste est affichée. Cette liste est constituée par des exercices littéraux. Ce sont aussi des exercices basés sur les renseignements que possède l'ordinateur à votre sujet.

Exercice "du prof"

Seuls les exercices du professeur qui ne font pas partie des exercices déjà vus d'une manière ou d'une autre sont affichés. Le déroulement est ensuite le même que pour le point précédent.

La fenêtre Historique

Cette fenêtre accumule les exercices accompagnés des commentaires et cela durant toute la session. Il est ensuite possible de l'imprimer pour en conserver une trace. Voici un exemple :



4. Consulter

Vous pouvez à tout moment consulter trois choses :

- les formules
  Vous déroulez le menu "Divers" et pointez Aide. Une fenêtre avec ascenseur vous permet de voir la liste des principales formules.
- un rapport.
  Ce rapport vous est propre. et est un check-up de vos connaissances.
- le noyau.
  Il vous montre le "noeud" de vos faiblesses. Ce sont les expressions présente dans au moins trois autres et que vous ne maîtrisez pas bien.

5. Changer d'utilisateur

Sans devoir quitter le logiciel, on peut changer d'utilisateur. Une série d'opérations est alors exécutée par l'ordinateur.

- sauver le fichier de l'élève
- effacer le contenu des fenêtres
- initialiser une série de choses
- demander l'identité du nouvel utilisateur.

6. Quitter

On quitte le logiciel et le fichier de l'élève qui demande à quitter est sauvé.

Septième partie

Conclusions

1. La réalisation

Nous avons défini le projet dans la seconde partie. Nous avons imaginé une stratégie pour offrir à l'élève la possibilité de progresser. La philosophie de cette stratégie est qu'il faut à tout moment tenir compte de l'état d'avancement des connaissances de l'élève et rejoindre ainsi l'esprit de l'évaluation formative. L'élève peut mettre le doigt sur le nœud de ses difficultés avec l'aide de l'ordinateur. Les commentaires aident à les comprendre. Il peut enfin faire des liens avec des formules et des expressions littérales. Ce logiciel réalise ce qui me semble presque impossible en pratique dans une classe : un suivi tout à fait personnalisé avec une attention toute particulière au sens des opérations, aux explications et à l'abstraction.

Je rappelle que seule une part a été réalisée et que ce didacticiel est un laboratoire. Malheureusement la phase d'expérimentation n'a pas eu lieu. Cette expérimentation pourrait aider à savoir si le fait que l'informaticien et l'enseignant ne font qu'une seule et même personne est un élément positif ou non.

2. Apport personnel

Ce travail a été d'une très grande richesse.

Tout d'abord, c'est la première fois que j'ai eu l'occasion de réaliser seul un travail complet depuis la conception jusqu'à l'implémentation. Cela m'a permis d'avoir une idée des difficultés de tous ordres que l'on rencontre dans ce genre de travail.

Ensuite, j'ai vu la difficulté qu'il y a parfois à rendre explicites des connaissances implicites. Quand il s'agit de faire faire une démarche par l'ordinateur il est nécessaire de faire émerger des mécanismes enfouis et qu'il n'avait jamais été utile d'examiner de plus près. Je pense, par exemple, aux fausses règles et à la manière dont les jugements récents effacent peu à peu les jugements anciens.

Enfin, j'ai vraiment apprécié de programmer dans un langage logique. Bien que déroutant au début, il m'a finalement procuré beaucoup de satisfaction.

Il reste tellement à faire, que j'ai dû fournir un réel effort pour arrêter afin de rédiger ce mémoire. En bref, le sujet m'a plu.

3 Perspectives

J'ai déjà cité une série de points dans la seconde partie qui me semblaient pouvoir faire l'objet de développements futurs. Pour rappel ces points sont :

- introduire la possibilité d'avoir des énoncés dans lesquels il pourrait y avoir des puissances.

- stocker les confusions d'un élève. On aurait ainsi la possibilité de les faire afficher. Mais cela pourrait servir également à rendre plus performante la recherche directe d'une erreur.
- faire apparaître la sémantique des règles. Nous avons déjà développé ce thème.
- remplacer une sélection par une autre expression plus concise et non obligatoirement par un entier ou un nombre fractionnaire
- donner la possibilité de réaliser des sélections multiples.
- présenter à l'élève, sans qu'il l'ait demandé, des exercices, obligatoires ou non, à résoudre.
- dans le mode de décomposition manuelle, permettre de faire une modification, dans une ligne quelconque, et voir la répercussion de cette modification dans les lignes suivantes.
- protéger l'accès au paramètrage par un mot de passe.
- créer une série d'utilitaires pour le professeur dont des fonctions statistiques.

Un des points le plus important me semble être d'aller plus loin dans la démarche manuelle par des sélections multiples et la possibilité de remplacer une expression par une autre plus concise.

Il y a encore pas mal d'idées à creuser. En voici quelques-unes.

- Super rapport.

On pourrait imaginer, à partir d'un ensemble de fichiers d'élèves, de faire un super rapport général qui permettrait d'avoir un véritable check up de la classe et ainsi de construire un cours sur mesure. Par exemple, les fichiers pourraient être fusionnés en un seul.

- Ecrire comme au tableau

Il serait bon de pouvoir rentrer les énoncés comme ils sont écrit au tableau en ce qui concerne les quotients. Cela évite de devoir placer des parenthèses supplémentaires.

- Attacher une connaissance à une famille

De la même manière que l'on a attaché une connaissance à une structure, on pourrait, après avoir défini un mode d'agrégation, attacher une connaissance aux familles.

- Générer des exercices

Il s'agirait ici de créer des exercices qui ne sont pas simplement des copies des structures rencontrées, mais des exercices plus complexes qui contiennent une ou plusieurs structures déjà rencontrées. On peut même envisager de pouvoir imprimer une liste d'exercices et créer ainsi une feuille de travaux.

- Générer une résolution contenant une erreur

Le professeur donnerait un énoncé, puis l'ordinateur générerait une résolution contenant une erreur typique de l'élève. La tâche de l'élève consisterait à localiser la ligne où se situe l'erreur pour la première fois dans la démarche descendante. Par exemple,

```
2*(3+7)-(5+6)+2*3
2*10-(5+6)+6
20-5+6+6
20+1+6
27
```

on attendrait de l'élève qu'il sélectionne -5+6.

- tester la distraction

Lorsqu'à plusieurs reprises, un élève commet des erreurs pour des structures qu'il connaît déjà bien, on pourrait attirer son attention sur ce constat. Ce pourrait être le point de départ d'une mesure de la distraction d'un élève pour une session d'exercices.

Pour ce qui est de perspectives plus larges,

• suggérer des démarches

Plaçons-nous dans le cadre de la démarche manuelle et de la possibilité de remplacer une expression par une autre plus concise. L'ordinateur, dans ses commentaires, pourrait suggérer des démarches. Je songe , par exemple, à des suggestions du type : il fallait mettre en évidence plutôt que de développer telle expression. Je pense spontanément à cette suggestion, car c'est, chez trop d'élèves, un réflexe qui leur porte préjudice. On développe sans se demander si ce développement est utile pour aboutir à l'objectif fixé.

• Expressions littérales

Je vois très bien aussi un didacticiel où on traiterait des expressions littérales au lieu d'expressions numériques. Ici bien sûr les réponses des élèves seraient obligatoirement des expressions littérales aussi. Plus concrètement, il s'agirait, par exemple, d'admettre des transformations successives du type :

```
2*a+5*b+4*(a+b*c)
2*a+5*b+4*a+4*b*c
6*a+b*(5+4*c)
```

• Dérivation d'expressions en x.

Avec les mêmes idées de structures et de connaissance, je vois aussi la possibilité d'un didacticiel qui serait une aide à la dérivation. Il y a des structures que l'on dériverait facilement et d'autres moins facilement. Ici aussi, il faut faire le lien entre la structure de l'expression que l'on dérive et les formules à appliquer.

Revenons à notre didacticiel.

• les ensembles flous

L'essai de localisation directe n'est pas très concluant. Le mode même de recherche basé sur la connaissance et des statistiques devrait être revu ou remanié. Nous sommes dans une démarche où une structure doit avoir été rencontrée suffisamment de fois par l'élève pour entrer en ligne de compte dans le processus de localisation directe. Ceci est une source de faiblesse de ce processus. Pourquoi ne pas remplacer cette notion de fréquence par une notion de possibilité ? Le professeur, qui joue le rôle de l'expert, a déjà dressé une liste de fausses règles. Allons plus loin. A chacune de ces fausses règles attachons une grandeur qui représenterait la possibilité d'être appliquée. En d'autres mots, créons un ensemble flou. Les notions de possibilité et de probabilité sont distinctes bien que toutes les deux mesurées par un nombre compris entre 0 et 1. En effet, on peut sentir la différence entre les deux notions en observant que ce qui est presque impossible est peu fréquent alors que ce qui est peu fréquent peut être fort possible. Un exemple. L'expression $\frac{a+c}{c} = a$ est rare et pourtant à haut risque ou à plus haut risque que -(a+b) =-a-b. Ce serait au professeur de déterminer ces coefficients de possibilité en tant qu'expert. En combinant les statistiques disponibles de l'élève et ce facteur de risque il serait peut-être possible d'augmenter la performance de la recherche. On peut trouver dans la littérature des applications des ensembles flous aux systèmes experts.

Nous aurions donc un ensemble flou qui serait l'ensemble des règles fausses à risque défini comme un ensemble de couples (règle fausse, μ) où μ est un coefficient d'appartenance à cet ensemble. Le nombre μ serait pris en compte dans les calculs conduisant à la recherche de l'expression ayant la plus grande possibilité d'être la source de l'erreur.

• pondération des erreurs

Nous avons mis toutes les erreurs dans le même panier. Mais nous pourrions bien imaginer de pondérer les erreurs. Une erreur de distraction du type opérations entre entiers pourrait avoir un coefficient d'importance différent de celui d'une erreur due à une fausse règle.

Dans le même esprit, si une erreur revient sans cesse, on pourrait augmenter le poids de cette erreur dans le calcul de la connaissance. Il me semble que faire cinq fois la même erreur de manière consécutive ou entrecoupée de bons résultats peut ne pas avoir la même signification.

4 Si c'était à refaire ...

• les fichiers texte des élèves

Sauvez les formes relatives à un élève peut se faire de deux manières : sous forme de fichier texte comme je l'ai réalisé ou sous forme de faits qui seront intégrés dans le didacticiel. La solution adoptée me semble bonne. N'importe quel petit traitement de texte peut les ouvrir. Le professeur a ainsi accès à toutes les données qui ont été sauvées avec notamment les compteurs attachés à chaque structure.

• la recherche de l'erreur

Pour ce qui est de la conception proprement dite, j'orienterais plus mon travail sur la décomposition manuelle qui, plus riche, se prête davantage à une approche qui tient compte de la démarche réelle de l'élève.

• des contre exemples

Je pense qu'il serait enrichissant d'adjoindre une manière supplémentaire d'attirer l'attention de l'élève en faisant appel à la technique des contre exemples. Bien choisis, ils peuvent être frappants. Considérons, par exemple, un élève qui fait l'erreur suivante $\frac{7}{8} + \frac{4}{9} = \frac{11}{17}$. Après lui avoir fait remarquer qu'il additionne les deux numérateurs et ensuite les deux dénominateurs, il est aisé de lui montrer sur un exemple évident que c'est faux. On pourrait attirer son attention sur le fait que $\frac{8}{2} + \frac{1}{1} \neq \frac{9}{3}$ car sinon 4+1=3. Ou encore on a $\frac{1}{1} + \frac{1}{1} \neq \frac{2}{2}$. Pour autant que les mêmes contre exemples soient utilisés systématiquement, on peut espérer qu'ils aident à éliminer l'usage des fausses règles. Ce qui est peut-être plus facile à faire apparaître, c'est la sémantique de ces cas particuliers. Dans le cas de l'exemple, une droite graduée suffit à faire apparaître que $\frac{8}{2} + \frac{1}{1} \neq \frac{9}{3}$.

• le choix de prolog

Si c'était à refaire je n'hésiterais pas à choisir de nouveau Prolog. Je puis l'affirmer car j'ai commencé à concevoir le didacticiel en Pascal. Rapidement je me suis rendu compte

qu'avec ce langage, le travail serait long et qu'en plus les changements éventuels étaient lourds à gérer. Je me suis donc orienté vers un autre langage après avoir fait une expérience concrète en langage procédural. Prolog me semblait très bien adapté au sujet. A titre indicatif, j'ai obtenu une diminution d'environ 80% de lignes de code en passant du Pascal à Prolog. Je ne peux pas passer sous silence la puissance de la grammaire DCG qui a permis de réaliser le contrôle syntaxique d'une expression rentrée au clavier. Au début de mon travail, j'ai fait un essai de tel contrôleur en Pascal. Je ne suis pas sûr qu'il m'aurait été possible d'en venir à bout sans y consacrer une énergie et un temps considérable.

Pour ce qui est du langage, le seul point sombre est que je n'ai pas eu l'occasion d'essayer un langage tel que Lisp qui aurait peut-être bien convenu également.

- le maquettage

Le maquettage me semble justifié. J'en veux pour preuve le fait que les idées de familles et de noyaux me sont venues à l'esprit durant la réalisation du travail. J'ai constaté que la liste des structures garnies de nombres qui était proposée à l'élève s'allongeait considérablement après quelques simulations. Cette longueur me semblait trop importante pour aider réellement l'élève à situer le noeud de ses problèmes. C'est seulement à partir de ce moment que j'ai réfléchi à une approche supplémentaire. Bien sûr, on peut toujours arguer du fait que l'on pouvait le prévoir, mais il faut aussi voir le monde tel qu'il est et non tel que l'on voudrait qu'il soit.

- globalement

Il est donc possible d'obtenir un diagnostic individualisé pour chaque élève. On pourrait générer des feuilles de travaux personnalisés sur base des données individuelles détenues par l'ordinateur. Il est même possible d'avoir des examens individualisés et permettre une progression différente d'un élève à l'autre.

Quant au choix du sujet, une seule phrase suffit : je ne regrette rien.

Index

Table des prédicats

Bibliographie

[1]. Farrenhy et Ghallab *Éléménts d'intelligence artificielle* Hermes pp9-10.
[2]. Meyer et Baudoin *Méthode de programmation* p279
[3]. Le Ministre de l'Education *Pour une école sans perdants.* 1964
[4]..G. De Landsheere *Dictionnaire de l'évaluation et de la recheche en éducation.*
[5]. M. De Montmollin *L'enseignement programmé* 1965, PUF
[6]. Shapiro *The Craft of Prolog*
[7]. Shapiro *The Art of Prolog.*
[8]. Closksin and Lellish. *Programming in Prolog*
[9]. M. Baron; R. Gras; J. Nicaud. *Environnements interactifs d'apprentissage avec ordinateur. Eyrolles*
[10] Sleeman and Brown. *Intelligent Tutoring Systems.* Academic Pres

Le code

Remarque

La manière dont on écrit le code évolue en fonction du degré de maîtrise que l'on a du langage utilisé. Il est clair que la partie écrite au début du travail pourrait être réécrite d'une manière plus concise.

Le code qui suit n'a pas été remanié. Il permet ainsi de conserver l'évolution de l'écriture. C'est ainsi que l'opérateur -> n'a été utilisé que tardivement.

```
/* ****************************************************************

            TABLE DES MATIERES

****************************************************************

exercices(personnel)
exercices(du_professeur)
R(recherche interactive)
D(Par l'ordinateur)
Divers(rapport_général)
Général(nouvel_utilisateur)
<ABOUT>(X)
presentation
mark_unmark(R, recherche auto)
bselection(A, Dest)
cselection
nonvide(Lf, Ll)

************** Fin de la table des matières ************************* */

/*
*********************************************************************

   ACTION DES MENUS

*********************************************************************

---------------------------------------------------------------------
MENU EXERCICES
---------------------------------------------------------------------
*/

'exercices'(personnel) :-
creer_exercice_pers(X),
(marked_item('Décomposition','Par l''ordinateur') ->
(solutiong(X),
enregistrerfaux,
erreur,
'exercices'(personnel));
bselection(X,personnel)).



'exercices'(à_travailler) :-
creer_exercice_trav(F),
(marked_item('Décomposition','Par l''ordinateur') ->
(solution(F),
enregistrerfaux,
erreur,
'exercices'(à_travailler));
bselection(F,à_travailler)).



'exercices'(du_professeur) :-
creer_exercice_prof(X),
(marked_item('Décomposition','Par l''ordinateur') ->
(solution(X),
enregistrerfaux,
erreur,
'exercices'(du_professeur));
bselection(X,du_professeur)).
```

```
'exercices'(mes_bases) :-
creer_exercice_mes_bases(Csg),
(marked_item('Décomposition','Par l''ordinateur') ->
(solution(Csg),
enregistrerfaux,
erreur,
'exercices'(mes_bases));
bselection(Csg,mes_bases)).


/*
-------------------------------------------------------------
MENU RECHERCHE
-------------------------------------------------------------
*/

'Recherche'('recherche interactive') :-
mark_unmark('Recherche','recherche auto').


'Recherche'('recherche auto') :-
mark_unmark('Recherche','recherche auto').

/*
-------------------------------------------------------------
MENU DECOMPOSITION
-------------------------------------------------------------
*/

'Décomposition'('Par l''ordinateur') :-
mark_unmark('Décomposition','Par l''ordinateur').

'Décomposition'('Par moi') :-
mark_unmark('Décomposition','Par l''ordinateur').

/*
-------------------------------------------------------------
MENU DIVERS
-------------------------------------------------------------
*/

'Divers'(rapport_général) :- rapportg.
'Divers'('Le_noyau') :- voirnoyau.
'Divers'(aide) :-aide.
'Divers'(configuration) :- configuration.

/*
-------------------------------------------------------------
MENU GENERAL
-------------------------------------------------------------
*/

'Général'(nouvel_utilisateur) :-
globale(identifiant,Z),
not(Z=bidon),
banner(stockerdisk(Z),['Enregistrement en cours du fichier',Z]),
remember(listeexpression,[]),
identifier,
enable_menu('exercices'),
enable_menu('Divers'),!.

'Général'(nouvel_utilisateur) :-
remember(chparametres,non),
miseajour(identifiant,bidon),
assert(forme(bidon,1,c,d,e,f)),
```

```
effacerfenetres,
creerrapport,
creerhistorique,
creerevolution,
creertampon,
creertampon2,
remember(listeexpression,[]),
identifier,enable_menu('exercices'),enable_menu('Divers').

'Général'(quitter) :-
globale(identifiant,Z),
banner(stockerdisk(Z),['Enregistrement en cours du fichier',Z]),
windows(prog,Wlist),
sauver_source(Wlist),
halt.


/*
-------------------------------------------------------------
MENU ABOUT
-------------------------------------------------------------
Charger automatiquement à l'ouverture du logiciel
*/

'<ABOUT>'(X) :-presentation.


presentation :-
monmessage([' Mémoire de fin d''études présenté par ~M~M Ph. Nicolas ~M ~M en vue de l''
obtention du diplôme de ~M~M Licence et Maîtrise en informatique ~M ~M Promoteur : C.
Cherton~M~M Facultés Universitaires Notre-Dame de la Paix ~M~M Namur  1993-1994']),
monmessage(['Aide à l''étude des expressions numériques à coefficients entiers dans l''
enseignement secondaire']).



/*
-------------------------------------------------------------
BASCULEMENT D'UN ITEM A L'AUTRE
-------------------------------------------------------------
Recherche
*/

mark_unmark('Recherche','recherche auto') :-
not(marked_item('Recherche','recherche auto')),
mark_item('Recherche','recherche auto'),
unmark_item('Recherche','recherche interactive'),!.

mark_unmark('Recherche','recherche auto') :-
unmark_item('Recherche','recherche auto'),
mark_item('Recherche','recherche interactive').

/* -------------------------------------------------------------
Decomposition
*/

mark_unmark('Décomposition','Par l''ordinateur') :-
not(marked_item('Décomposition','Par l''ordinateur')),
mark_item('Décomposition','Par l''ordinateur'),
unmark_item('Décomposition','Par moi'),
enable_menu('Recherche'),!.

mark_unmark('Décomposition','Par l''ordinateur') :-
unmark_item('Décomposition','Par l''ordinateur'),
mark_item('Décomposition','Par moi'),
```

```
mark_item('Recherche','recherche interactive'),
unmark_item('Recherche','recherche auto'),
disable_menu('Recherche').

/*
-------------------------------------------------------------------------
SUITE DES PROCEDURES LIEES DIRECTEMENT AUX MENUS
-------------------------------------------------------------------------
*/

bselection(A,Dest) :-
remember(bval,[]),
remember(fval,[]),
effacertampon,
write('Tampon',A),
writeseqnl('Evolution',['Les évaluations successives :']),
voirevolution,
globale(expression,Eee),
remember(vexp,Eee),
mondialogue3(Bn,R,'résultat',Text,Exp),
effacerblanc,
wsearch('Tampon',Text,0,From,To),
cursor('Tampon',From,To),
clear('Tampon'),
cursor('Tampon',From,From),
/* --- mettre des parentheses quand le résultat est négatif --- */

(Exp>0 -> write('Tampon',Exp);
writeseq('Tampon',['(',Exp,')'])),
effacerblanc,
wlen('Tampon',Long),
wsltxt('Tampon',0,Long,T2),
cursor('Historique',-1,-1),
writeseqnl('Historique',[T2]),
remember(vexp,T2),
((pname(T2m,T2),(transfo2(T2m);fractionentiere(T2m))) ->
 (message(['Terminé !!']),voirerreur,effacertampon,!);
cselection,
'exercices'(Dest)).


/* ========================================================================== */


cselection:-

mondialogue3(Bn,R,'résultat',Text,Exp),
effacerblanc,
wsearch('Tampon',Text,0,From,To),
cursor('Tampon',From,To),
clear('Tampon'),
cursor('Tampon',From,From),


/* --- mettre des parentheses quand le résultat est négatif --- */

(Exp>0 -> write('Tampon',Exp);
writeseq('Tampon',['(',Exp,')'])),
effacerblanc,
wlen('Tampon',L),
wsltxt('Tampon',0,L,T2),
remember(vexp,T2),
cursor('Historique',-1,-1),
writeseqnl('Historique',[T2]),
((pname(T2m,T2),(transfo2(T2m);fractionentiere(T2m))) -> (message(['Terminé !!']),
```

```
voirerreur,voirconfusions,
 effacertampon,!);
cselection).

/* ------------------------------------------------------------------------ */

nonvide(Lf,Ll) :-
Lf=[],Ll=['Choisir un exercice personnel ou du prof'],!.
nonvide(Lf,Ll) :-
Ll=Lf.
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

erreur
fauteindet(X)
analyse_erreur(X)
afficherformule(X)
maide(a*(b+c), On a a*(b+c)=a*b+a*c. C'est la distributivité)
listeerreur(Ef)
voirerreur
listevalfausse(Ef)
lerreur(X)
trouverlg(X)

************** Fin de la table des matières ************************ */

/*
*****************************************************************************

LES ERREURS

*****************************************************************************
-----------------------------------------------------------------------------
APPEL PRINCIPAL.  RECHERCHE LA FORMULE MAL APPLIQUEE
-----------------------------------------------------------------------------
X est l'expression fausse où naît l'erreur
*/

erreur:-
globale(faux,X),
fauteindet(X),
ssimp(X,Xs,2),
remember(lgarnir,[a,b,c,d]),
sgarnir(Xs,Xss),
pname(Xss,Xsss),
afficherformule(Xss),!.

erreur.

/*
-----------------------------------------------------------------------------
AIGUILLAGE SUIVANT QUE L'ON A DETECTE LE TYPE DE FAUTE OU NON
-----------------------------------------------------------------------------
lance une petite analyse de l'erreur si on n'a pas trouvé de fausses règles
*/

fauteindet(X) :-
recall(trouverfaute,non),  % initialisé dans le predicat "faute"
analyse_erreur(X),!.

fauteindet(X) :- remember(afficherform,oui).

/*
-----------------------------------------------------------------------------
MESSAGE ADEQUAT
-----------------------------------------------------------------------------
*/

analyse_erreur(X) :-
functor(X,_,N),
X=..[Op,G,D],
integer(G),
```

```
integer(D),
remember(afficherform,non),
writeseqnl('Historique',['Une simple erreur de calcul, car une opération élémentaire
entre entiers.']),!.

analyse_erreur(X) :-
writeseqnl('Historique',['Peut-être une simple erreur de calcul.']),
remember(afficherform,non).


/* --- recherche de la formule mal appliquée --- */

afficherformule(X) :-
recall(afficherform,oui),
maide(X,Mes),
writeseqnl('Historique',['~M','Votre formule mal appliquée : ','~M',Mes]),!.


/* --- si on n'a pas trouvé de formule dans la liste des messages --- */

afficherformule(X).


/*
-------------------------------------------------------------------------
LISTE DES MESSAGES DE FORMULES
-------------------------------------------------------------------------
*/

maide(a*(b+c),'On a a*(b+c)=a*b+a*c. C''est la distributivité'):-!.
maide(a*(b-c),'On a a*(b-c)=a*b-a*c. C''est la distributivité') :- !.
maide(a*(b*c),'On a a*(b*c)=a*b*c.') :- !.
maide(a*(b/c),'On a a*(b/c)=(a*b)/c. La multiplication s''applique au numérateur
uniquement') :- !.

maide((a+b)*c,'On a (a+b)*c=a*c+b*c. C''est la distributivité') :- !.
maide((a-b)*c,'On a (a-b)*c=a*c-b*c. C''est la distributivité') :- !.
maide((a*b)*c,'On a (a*b)*c=a*b*c.') :- !.
maide((a/b)*c,'On a (a/b)*c=(a*c)/b. La multiplication s''applique au numérateur
uniquement') :- !.

maide(a-(b+c),'On a a-(b+c)=a-b-c. En supprimant la parenthèse, appliquer la règle des
signes') :- !.
maide(a-(b-c),'On a a-(b-c)=a-b+c. En supprimant la parenthèse, appliquer la règle des
signes') :- !.

maide((a+b)*(c+d),'On a (a+b)*(c+d)=a*c+a*d+b*c+b*d. C''est la distibutivité') :- !.
maide((a+b)*(c-d),'On a (a+b)*(c-d)=a*c-a*d+b*c-b*d. C''est la distibutivité') :- !.
maide((a-b)*(c+d),'On a (a-b)*(c+d)=a*c+a*d-b*c-b*d. C''est la distibutivité') :- !.
maide((a-b)*(c-d),'On a (a-b)*(c-d)=a*c-a*d-b*c+b*d. C''est la distibutivité') :- !.

maide(a+b/c,'On a a+b/c=(a*c)/c+b/c=(a*c+b)/c. C''est la réduction,au même
dénominateur') :- !.
maide(a-b/c,'On a a-b/c=(a*c)/c-b/c=(a*c-b)/c. C''est la réduction,au même
dénominateur') :- !.
maide(a/b+c,'On a a/b+c=a/b+(c*b)/b=(a+c*b)/b. C''est la réduction,au même
dénominateur') :- !.
maide(a/b-c,'On a a/b-c=a/b-(c*b)/b=(a-c*b)/b. C''est la réduction,au même
dénominateur') :- !.

maide(a*b/c, 'On a a*b/c=(a*b)/c=(a/c)*b=a*(b/c)') :- !.
maide((a/b)/c, 'On a (a/b)/c=a/(b*c)') :- !.
maide(a/(b/c), 'On a a/(b/c)=(a*c)/b') :- !.

maide(a/b+c/d, 'On a a/b+c/d=(a*d+b*c)/(b*d)') :- !.
```

```
maide(a/b-c/d, 'On a a/b-c/d=(a*d-b*c)/(b*d)') :- !.
maide((a/b)*(c/d), 'On a (a/b)*(c/d)=(a*c)/(b*d)') :- !.
maide((a/b)/(c/d), 'On a (a/b)/(c/d)=(a*d)/(b*c)') :- !.

maide(X,'Pas de règle particulière').


/*
*****************************************************************************

DES ERREURS LIEES A LA RECHERCHE NON AUTOMATIQUE

*****************************************************************************
-----------------------------------------------------------------------------
Enregistrement en mémoire d'une nouvelle expression mal évaluée ou
non évaluée
-----------------------------------------------------------------------------
Les expressions en question sont rangées dans une liste
Une nouvelle expression est placée en tête de liste
 */


listeerreur(Ef) :-
recall(expfausse,L),
Nl=[Ef|L],
remember(expfausse,Nl),!.

listeerreur(Ef) :-
remember(expfausse,Ef).

/*
-----------------------------------------------------------------------------
Appel de la liste des expressions sans évaluation correcte
-----------------------------------------------------------------------------
Dans tous les cas les deux listes (liste des expressions et liste des valeurs sont
renversées !
*/

/* --- il y a effectivement une erreur --- */

voirerreur :-
/* --- renversement des liste pour faire apparaître les ererurs dans
       le même ordre que celui où elles ont été rencontrées --- */
recall(expfausse,Ll),reverse(Ll,Lg),remember(expfausse,Lg),
recall(valfausse,Kl),reverse(Kl,Kg),remember(valfausse,Kg),

recall(expfausse,L),
not(L=[]),
writeseqnl('Historique',['Voici la liste des expressions sans évaluation correcte :']),
remember(lgexp,0),
write(L),
map(trouverlg,L), % pour l'alignement à l'écran
map(lerreur,L),!.

/* --- il n'y a pas d'erreur --- */

voirerreur :-
writeseqnl('Historique',['Bravo. Vous n''avez fait aucune erreur']).

/*
-----------------------------------------------------------------------------
Enregistrement en mémoire de la liste des valeurs erronnées
-----------------------------------------------------------------------------
Rappel : quand on a renoncé a évalué une expression, c'est $ que l'on enregistre en
lieu et place d'une valeur numérique
```

```
*/

/* --- l'expression fausse n'est pas la première de l'exercice --- */

listevalfausse(Ef) :-
recall(valfausse,L),
Nl=[Ef|L],
remember(valfausse,Nl),!.

/* --- l'expression fausse est la première de l'exercice --- */

listevalfausse(Ef) :-
remember(valfausse,Ef).


/*
-------------------------------------------------------------------------
Affichage de la liste des expressions sans évaluation correcte
-------------------------------------------------------------------------
*/

lerreur(X) :-
afficher(X,V),
recall(valfausse,L),
L=[T|Q],
pname(X,Xe),
recall(lgexp,Lg),
Lgg is - (Lg + 3),
fw('Historique',[c(Lgg)],[Xe]),
(not(T= $) ->writeseqnl('Historique',['Vous l''avez évaluée à ',T, ' au lieu de ',V]);
(writeseqnl('Historique',['Vous avez renoncé à l''évaluer']))),
remember(valfausse,Q).

/*
-------------------------------------------------------------------------
Regle l'affichage des commentaires liés aux expression mal évaluées ou sans évaluation
-------------------------------------------------------------------------
On stocke dans Lg la plus grande longueur rencontrée en calculant la longueur
de chacune des expressions qui sera affichée
*/


trouverlg(X) :-
pname(X,Xp),
stringof(S,Xp),
length(S,Lg),
writenl(Lg),
recall(lgexp,Lgvieux),
(Lg>Lgvieux ->remember(lgexp,Lg);true).
```

```
/*

VOICI LES PRINCIPALES FORMULES



a-(b+c)=a-b-c
a-(b-c)=a-b+c
a+(b-c)=a+b-c
a+(b+c)=a+b+c

a*(b+c)=a*b+a*c
a*(b-c)=a*b-a*c
(a+b)*c=a*c+b*c
(a-b)*c=a*c-b*c

(a+b)*(c+d)=a*c+b*c+a*d+b*d
(a-b)*(c+d)=a*c-b*c+a*d-b*d
(a+b)*(c-d)=a*c+b*c-a*d-b*d
(a-b)*(c-d)=a*c-b*c-a*d-b*d


a+b/c=(a*c)/c+b/c = (a*c+b)/c
a-b/c=(a*c)/c-b/c = (a*c-b)/c
a*b/c=a*(b/c) = (a*b)/c
(a/b)/c = a/(b*c)
a/(b/c) = (a*c)/b

a/b+c = a/b+(c*b)/c = (a+c*b)/c
a/b-c = a/b-(c*b)/c = (a-c*b)/c
a/b+c/d = (a*d+b*c)/(b*d)
a/b-c/d = (a*d-b*c)/(b*d)
(a/b)*(c/d) = (a*c)/(b*d)
(a/b)/(c/d) = (a*d)/(b*c)
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

recherchefaute(X, Sortie)
faute(Entree, Sortie, Valeur, M, A, B, C, D)
entree(Entree, A, B, C, D, T, M)
sortie(divientier, A, B, C, rien, S)
messageerreur(N, A, B, C, D, S)
operationadd(A, B, A+B)
operationmul(A, B, A*B)
operationdivi(A, B, A/B)
listeconfusions(faute(E, S, V, N, A, B, C, D))
voirconfusions
lireconfusions(X)

************** Fin de la table des matières ************************* */


/*
-------------------------------------------------------------------------
 ESSAI DE DETECTION DU TYPE DE FAUTE QUAND IL Y A FAUTE
-------------------------------------------------------------------------
Ce but réusit toujours
A l'aide de la réponse fausse (Repfausse) proposée par l'élève, on
recherche l'expression (Sortie) qui y a conduit en lieu et place de
l'expression à évaluée (X).
*/

recherchefaute(X,Sortie) :-
globale(reponse,Repfausse),
pname(Toto,Repfausse),
Toto1 is Toto,
faute(X,Sortie,Toto1,N,A,B,C,De),
not(Sortie=non),
pname(Sortie,S),
concat(['Avez vous fait  : ',S,' ?'],Msgerreur),
/* --- on place le dialogue en cours (n°4) dans sa seconde fonction
en changeant la sémantique des boutons --- */
setditem('Valeur',1,'oui'),
setditem('Valeur',6,'non'),
setditem('Valeur',3,Msgerreur),
remember(expfausse,[N,A,B,C,De,Sortie]),!,fail.



/* --- si pas de faute --- */

recherchefaute(X,Sortie).


/*
-------------------------------------------------------------------------
RECHERCHE D'UNE FAUSSE REGLE
-------------------------------------------------------------------------
Voir documentation annexe
Ce but réussit toujours
Entree est la structure de l'expression
A,B,C,D sont les composante de la structure
Type est le type de la famille de la structure
M est le numéro du type de structure considérée
Sortie est l'expression fausse qui conduit au résultat de l'élève
```

```
exemple :

faute(5*(2+1),F,11,A,B,C,D,E) conduit à l'unification de F avec 5*2+1
*/

/* --- la recherche réussit --- */

faute(Entree,Sortie,Valeur,M,A,B,C,D) :-
entree(Entree,A,B,C,D,Type,M),
sortie(Type,A,B,C,D,Sortie),
afficher(Sortie,Valeur1),
Valeur is Valeur1,
remember(trouverfaute,oui),!.

/* --- la recherche a échoué --- */

faute(Entree,non,Valeur,M,A,B,C,D) :- remember(trouverfaute,non).

/*
-------------------------------------------------------------------------
LES FAUSSES REGLES
-------------------------------------------------------------------------
*/

entree(Entree,A,B,C,D,T,M) :- Entree=A*(B+C),T=mul,D=rien,M=1.
entree(Entree,A,B,C,D,T,M) :- Entree=A*(B-C),T=mul,D=rien,M=2.
entree(Entree,A,B,C,D,T,M) :- Entree=(A+B)*C,T=mul,D=rien,M=3.
entree(Entree,A,B,C,D,T,M) :- Entree=(A-B)*C,T=mul,D=rien,M=4.

entree(Entree,A,B,C,D,T,M) :- Entree=A-(B-C),T=add,D=rien,M=5.
entree(Entree,A,B,C,D,T,M) :- Entree=A-(B+C),T=add,D=rien,M=6.
entree(Entree,A,B,C,D,T,M) :- Entree=A+(B-C),T=add,D=rien,M=7.
entree(Entree,A,B,C,D,T,M) :- Entree=A+(B+C),T=add,D=rien,M=8.

entree(Entree,A,B,C,D,T,M) :- Entree=A/B+C,integer(C),T=divientier,D=rien,M=9.
entree(Entree,A,B,C,D,T,M) :- Entree=A/B-C,integer(C),T=divientier,D=rien,M=10.
entree(Entree,A,B,C,D,T,M) :- Entree=(A/B)*C,integer(C),T=divientier,D=rien,M=11.
entree(Entree,A,B,C,D,T,M) :- Entree=(A/B)/C,integer(C),T=divientier,D=rien,M=12.

entree(Entree,A,B,C,D,T,M) :- Entree=(A/B)+(C/D),T=dividivi,M=13.
entree(Entree,A,B,C,D,T,M) :- Entree=(A/B)-(C/D),T=dividivi,M=14.


sortie(divientier,A,B,C,rien,S) :-
((operationadd(A,C,R1),S=R1/B);
(operationmul(A,C,R1),S=R1/B);
(operationmul(A,C,R1),operationmul(B,C,R2),S=R1/R2);
(operationmul(B,C,R1),S=A/R1)).


sortie(dividivi,A,B,C,D,S) :-
operationmul(A,C,R1),operationmul(B,D,R2),S=R1/R2.
sortie(dividivi,A,B,C,D,S) :-
operationadd(A,C,R1),operationadd(B,D,R2),S=R1/R2.

sortie(mul,A,B,C,rien,S) :-
(operationmul(A,B,R1),operationadd(R1,C,S));
(operationmul(A,C,R1),operationadd(R1,B,S)).

sortie(add,A,B,C,rien,S) :-
operationadd(A,B,R1),operationadd(R1,C,S).

/* --- liste des messages d'erreur --- */

messageerreur(N,A,B,C,D,S) :-
```

```
((N=1->(Op1='*',Op2='+'));
(N=2->(Op1='*',Op2='-'))),
afficher(Op1(A,Op2(B,C)),E),
writeseqnl('Historique',['Vous confondez ',A,Op1,'(',Op2(B,C),') avec',S]),
writeseqnl('Historique',['On a',A,Op1,'(',Op2(B,C),')=',A*B,Op2,A*C,'=',E]).


messageerreur(N,A,B,C,D,S) :-
((N=5->(Op1='-',Op2='-'));
(N=6->(Op1='-',Op2='+'));
(N=7->(Op1='+',Op2='-'));
(N=8->(Op1='+',Op2='+'))),
functor(S,F2,P2),
(F2='-'->Op='+';Op='-'),
afficher(Op1(A,Op2(B,C)),E),
writeseqnl('Historique',['Vous confondez ',A,Op1,'(',Op2(B,C),') avec',S]),
writeseqnl('Historique',
['On a',A,Op1,'(',Op2(B,C),')=',A,Op1,B,Op,C,'=',E]).


messageerreur(N,A,B,C,D,S) :-
((N=3->(Op1='+',Op2='*'));
(N=4->(Op1='-',Op2='*'))),
afficher(Op2(Op1(A,B),C),E),
writeseqnl('Historique',['Vous confondez (',Op1(A,B),')','*',C,' avec',S]),
writeseqnl('Historique',['On a (',Op1(A,B),')','*',C,'=',*(A,C),Op1,*(B,C),'=',E]).

messageerreur(N,A,B,C,D,S) :-
((N=13,Op='+');(N=14,Op='-')),
afficher(Op(A/B,C/D),E),
writeseqnl('Historique',['Vous confondez (',A/B,')',Op,'(',C/D,') avec',S]),
writeseqnl('Historique',['On a', A/B,Op,C/D,'=(',A*D,Op,C*B,')/(',B*D,')=',E]).

messageerreur(N,A,B,C,D,S) :-
((N=9->Op='+');
(N=10->Op='*')),
afficher(Op(A/B,C),E),
writeseqnl('Historique',['Vous confondez ',A/B,Op,'(',B,') avec',S]),
writeseqnl('Historique',['On a', A/B,Op,C,'=',A/B,Op,'(',B*C,')',/,B,'=',
'(',A+C*B,')',/,B,'=',E]).


operationadd(A,B,+(A,B)).
operationadd(A,B,-(A,B)).
operationmul(A,B,*(A,B)).
operationdivi(A,B,/(A,B)).


/*
-------------------------------------------------------------------------
DES CONFUSIONS
-------------------------------------------------------------------------
Mémorisation des confusions à l'aide de deux listes.
Une première liste est constituée des expressions à évaluées.
Une seconde liste est celle des expressions évaluées à la place des correctes
*/

/* construction des listes */
listeconfusions(faute(E,S,V,N,A,B,C,D)) :-
not(S=non),
recall(bval,Lb),
Nlb=[E|Lb],remember(bval,Nlb),
recall(fval,Lf),
Nlf=[S|Lf],remember(fval,Nlf),!.
```

```
listeconfusions(faute(E,S,V,N,A,B,C,D)).

/* préparation pour la lecture */

voirconfusions :-
recall(bval,Lb),
not(Lb=[]),
map(lireconfusions,Lb),!.

voirconfusions.

/* lecture proprement dite des confusions */

lireconfusions(X) :-
recall(fval,L2),
L2=[T2|Q2],
writeseqnl('Historique',['Vous confondez peut être ',X, 'avec',T2]),
remember(fval,Q2).
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

init_exercices(Dest)
creer_exercice_trav(F)
creer_exercice_pers(X)
creer_exercice_prof(X)
simp(E, __1)
s(*, X, Y, X*Y)
garnir(E, N)
g(*, X, Y, X*Y)
sgarnir(E, N)
ssimp(E, __1, N)
listeformes(Lf)
listeformes(L, Lsd)
decapiter(E, F)
garnirlettre(E, F)
mettreforme(L, F)
longueur(A, L)
defaire(A, B)
keepAfter([Char| Res], [Char], Res)
selectionfaiblesses(Critere, A)

************** Fin de la table des matières ************************** */


/*
*****************************************************************************

CREATION DES EXERCICES

*****************************************************************************
*/

init_exercices(Dest) :-
remember(expfausse,[]),
remember(valfausse,[]),
effacerevolution,
effacerfauxvrai,
miseajour(origine,Dest).



creer_exercice_trav(F) :-
init_exercices(ordi),
criterefaiblesse(Critere),
findall(A,selectionfaiblesses(Critere,A),L),
keysort(L,Ls,-1),
map(mettreforme,Ls,Lf),
nonvide(Lf,Ll),
essai(Choix,Ll,'à travailler ... '),
not(Choix=[]),
Choix=[C],
defaire(C,Cd),
pname(F,Cd),
miseajour(expression,Cd),   % enregistrer un atome
garnirhistorique,
souvenir.

creer_exercice_pers(X) :-
init_exercices(pers),
```

```
dialogueexp(Btn,Rap),  % Rap n'est pas utilisé;expression est mise a jour
garnirhistorique,
globale(expression,A),
pname(X,A),
souvenir.

creer_exercice_prof(X) :-
init_exercices(prof),
open(prof),
chargerprof([],S),
close(prof),
map(pname,S,L),
essai(Choix,L,'Suggestions ... '),
Choix=[D],
miseajour(expression,D),  % enregistrer un atome
garnirhistorique,
pname(X,D).

creer_exercice_mes_bases(Csg) :-
init_exercices(ordi),
listeformes(L),
map(pname,L,L2),
nonvide(L2,Li),
essai(Choix,Li,'Les bases ...'),
not(Choix=[]),
Choix=[C],
pname(Ca,C),
simp(Ca,Cs),
garnir(Cs,Csg),
write(Csg),
pname(Csg,Cp),
miseajour(expression,Cp),   % enregistrer un atome
garnirhistorique,
souvenir.
/*
*****************************************************************************

GESTION DES ARBRES

*****************************************************************************
-----------------------------------------------------------------------------
 PASSAGE D'UNE EXPRESSION NUMÉRIQUE OU LITTERAL À SA STRUCTURE
-----------------------------------------------------------------------------
dépouille un arbre de ses nombres pourconserver uniquement la structure
exmple : E=4+6*(1-5) et F=_+_*(_-_)
*/

simp(E,_) :- integer(E),!.
simp(E,_) :- on(E,[a,b,c,d,e,f,g]),!.
simp(E,F) :-
E=..[Op,La,Ra],
simp(La,X),
simp(Ra,Y),
s(Op,X,Y,F).
simp(-E,F):- simp(E,G),F = -G.


s('*',X,Y,X*Y).
s('-',X,Y,X-Y).
s('+',X,Y,X+Y).
s('/',X,Y,X/Y).

/*
-----------------------------------------------------------------------------
GARNI UNE STRUCTURE AVEC DES ENTIERS ALEATOIRES
```

```
-------------------------------------------------------------------------------
exemple : E=_+_*(_-_) et  F=4+6*(1-5)
F est un terme composé
*/

garnir(E,N) :- var(E),random(N),!.
garnir(E,F) :-
E=..[Op,La,Ra],
garnir(La,X),
garnir(Ra,Y),
g(Op,X,Y,F),!.
garnir(-E,N) :- garnir(E,G),N = -G.


g('*',X,Y,X*Y).
g('-',X,Y,X-Y).
g('+',X,Y,X+Y).
g('/',X,Y,X/Y).

/*
-------------------------------------------------------------------------------
GARNI UNE STRUCTURE AVEC DES LETTRES
-------------------------------------------------------------------------------
Les lettres sont dans la liste L qui est amputée de sa tete
chaque fois qu'une lettre a été utilisée; Ainsi, chaque lettre
n'est utilisée qu'une et une seule fois
exemple :
E=_+_*(_+_)
F=a+b*(c+d)
précondition : E ne peut avoir plus de deux niveaux
*/

sgarnir(E,N) :- var(E),recall(lgarnir,L),L=[N|Q],remember(lgarnir,Q),!.
sgarnir(E,F) :-
E=..[Op,La,Ra],
sgarnir(La,X),
sgarnir(Ra,Y),
g(Op,X,Y,F),!.
sgarnir(-E,N) :- sgarnir(E,B),N = -B.




/*
-------------------------------------------------------------------------------
DONNE LA STRUCTURE D'UNE EXPRESSION EN NE RETENANT
QUE LES N PREMIERS NIVEAUX
-------------------------------------------------------------------------------
E est un terme composé. Ici une expression numérique
N est le nombre de niveaux a retenir
F est la structure renvoyée
exemple
ssimp(4+3*(1+2/3),F,2) donne après unification :F = _+_*_
en réalité, ici on ne l'utilise qu'avec N=2
*/

ssimp(E,_,N) :- integer(E),!.
ssimp(E,F,0) :- !.
ssimp(E,F,N) :-
E=..[Op,G,D],
N1 is N-1,
ssimp(G,X,N1),
N2 is N-1,
ssimp(D,Y,N2),
```

```
s(Op,X,Y,F),!.
ssimp(-E,F,N) :- ssimp(E,G,N),F = -G.


/*
-------------------------------------------------------------------
 LISTE DES FAMILLES = DRESSER LA LISTE DES "FORMES" SANS DOUBLE
-------------------------------------------------------------------
*/
/* --- Lf est la liste des formes sans double --- */
listeformes(Lf) :-
criterefaiblesse(Cr),
findall(A,(forme(A,C,_,_,_,_),C<Cr),L),
map(garnir,L,L5),
% garnir car ssimp travaille sur du garni
map(decapiter,L5,L2),
map(garnirlettre,L2,L3),
killdouble(L3,Lf),!.

/* --- liste des formes (L) et cette même liste sans double (Lsd) --- */

listeformes(L,Lsd) :-
criterefaiblesse(Cr),
findall(A,(forme(A,C,_,_,_,_),C<Cr),L),
map(garnir,L,L5),
% garnir car ssimp travaille sur du garni
map(decapiter,L5,L2),
map(garnirlettre,L2,L3),
killdouble(L3,Lsd),!.


decapiter(E,F) :-
ssimp(E,F,2).

garnirlettre(E,F) :-
remember(lgarnir,[a,b,c,d]),
sgarnir(E,F).


/*
*************************************************************************

UTILITAIRES POUR LA LISTE DES EXPRESSIONS DONT LA CONNAISSANCE EST "PETITE"

*************************************************************************
*/

mettreforme(L,F) :-
L=A-B,
longueur(A,4),garnir(B,Bg),pname(Bg,Bgp),concat(A,' : ',Ac),concat(Ac,Bgp,F),!.

mettreforme(L,F) :-
L=A-B,
longueur(A,3),garnir(B,Bg),pname(Bg,Bgp),concat(A,'   : ',Ac),concat(Ac,Bgp,F),!.

mettreforme(L,F) :-
L=A-B,
longueur(A,1),garnir(B,Bg),pname(Bg,Bgp),concat(A,'     : ',Ac),concat(Ac,Bgp,F),!.

/* ------------------------------------------------------------------- */

longueur(A,L) :- name(A,J),length(J,L).
/* ------------------------------------------------------------------- */
```

```
defaire(A,B):-
  name(A,As),
  keepAfter(As,":",Bs),
  name(B,Bs).

keepAfter([Char|Res],[Char],Res).
keepAfter([C|Q],[Char],Res):-
  keepAfter(Q,[Char],Res).

/* ------------------------------------------------------------------------ */


selectionfaiblesses(Critere,A) :-
forme(X,Con,_,_,_,_),
Critere>Con,
Ncon is Con*100,
int(Ncon,Nncon),
Nvcon is Nncon/100,
A=Nvcon-X.
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

explorer2(X)
dialoguereponse(X, E, Rep)
mouinon(X, E, __1)
verificationresultat(D, X, E, Rep)
auto(E)
listessexp(E, L)
lss(X, Es)
nbrecas(X, N, Cb, Cf)
si(E, Sexp)
suiterep(Rep, E, L)
analyserep(E, ValE, Rep)
bonneselection2(E, Se)
divers(Dial, 1, R, Text, Exp)
syntaxerep2(D, 1, Entree, X, _1, Rep)

************** Fin de la table des matières ************************* */

/*
*****************************************************************************

EXPLORATION DE L'ARBRE D'UNE STRUCTURE NUMERIQUE

*****************************************************************************
-----------------------------------------------------------------------------
PROGRAMME PRINCIPAL
-----------------------------------------------------------------------------
Voir documentation annexe
Les valeurs de "globale" sont mises à jour dans la procédure attachée
au dialogue utilisé pour connaitre la réponse de l'utilisateur.
*/

explorer2(X) :-
not(integer(X)),not(fractionentiere(X)),afficher(X,E),
modeevolution(X,E),
dialoguereponse(X,E,Rep),
Rep=oui,!.

explorer2(-X) :-
explorer2(X).

explorer2(T) :- integer(T),miseajour(vrai,T).
explorer2(T) :- fractionentiere(T),miseajour(vrai,T).


explorer2(T1 + T2) :- globale(faux,T1+T2),explorer2(T1),fail.
explorer2(T1 + T2) :- globale(faux,T1+T2),globale(vrai,T1),explorer2(T2).

explorer2(T1 - T2) :- globale(faux,T1-T2),explorer2(T1),fail.
explorer2(T1 - T2) :- globale(faux,T1-T2),globale(vrai,T1),explorer2(T2).

explorer2(T1 * T2) :- globale(faux,T1*T2),explorer2(T1),fail.
explorer2(T1 * T2) :- globale(faux,T1*T2),globale(vrai,T1),explorer2(T2).

explorer2(T1 / T2) :- globale(faux,T1/T2),explorer2(T1),fail.
explorer2(T1 / T2) :- globale(faux,T1/T2),globale(vrai,T1),explorer2(T2).
/* cette dernière clause est atteinte quand on a affaire à un entier ou une
fractionentière. Elle force la reussite pour ne pas rendre la main au dialogue */
explorer2(Y).
```

```
/*
-------------------------------------------------------------------------
 CHOIX DU TYPE D'INTERACTION POUR LA REPONSE
-------------------------------------------------------------------------
*/

dialoguereponse(X,E,Rep) :-
reponseentree(oui),
mondialogue(Btn,R,'résultat',E,X,Rep),!.

dialoguereponse(X,E,Rep) :-
mouinon(X,E,Rep).

/*
-------------------------------------------------------------------------
DIALOGUE OUI-NON
-------------------------------------------------------------------------
*/

mouinon(X,E,_) :-
pname(E,Ep),
pname(X,Xp),
concat([Xp,=,Ep],Ser),
mdialog(210,350,70,60,[button(10,10,20,35,oui),button(40,10,20,35,non)],Btn2),
Btn2=1,
globale(origine,Orig),
enregistrer2(X,1,Orig),
miseajour(vrai,X),!.

mouinon(X,E,non) :-
miseajour(faux,X).

/*
-------------------------------------------------------------------------
VERIFIE SI UN RESULTAT ENTRE AU CLAVIER EST CORRECT OU NON
-------------------------------------------------------------------------
Ce but réussit toujours
*/

/* --- le résultat est correct --- */

verificationresultat(D,X,E,Rep) :-
globale(reponse,Exp),
pname(Rt,Exp),
Rt =:= E,
setditem(D,3,'bon'),
globale(origine,Orig),
enregistrer2(X,1,Orig),
miseajour(vrai,X),!,
Rep=oui,!.

/* --- le résultat est faux --- */

verificationresultat(D,X,E,Rep) :-
setditem(D,3,'faux'),beep(1),
miseajour(faux,X),Rep=non,
recherchefaute(X,Sortie).


/*
*************************************************************************

RECHERCHE AUTOMATIQUE DE L'ERREUR

*************************************************************************
```

```
------------------------------------------------------------------------
APPEL PRINCIPAL
------------------------------------------------------------------------
*/
/*
A MODIFIER
il faut avant de lancer la recherche que l'expression soit fausse !!
*/

auto(E) :-
afficher(E,ValE),
analyserep(E,ValE,Rep),
(Rep=oui -> !;
(listessexp(E,L),(not(L=[]))->(beep(20),suiterep(Rep,E,L));
(message(['Rien à suggerer.~M Passage en mode interactif']),effacerevolution,solution(
E)))).


/*
------------------------------------------------------------------------
LISTE APRES SELECTION
------------------------------------------------------------------------
recherche  la liste des expressions deja vues qui sont sous expression de l'expression
donnée et
qui satisfont a des criteres
E est une expression numérique (terme composé)
Es est la structure associée
L est la liste des structures déjà vues contenues ds E
X est une structure déjà vue
*/


listessexp(E,L) :-
simp(E,Es),
findall(X,lss(X,Es),L).


/*
------------------------------------------------------------------------
SELECTION DES STUCTURES
------------------------------------------------------------------------
en fct de la connaissance et du nombre de fois qu'elles ont été rencontrées
*/


lss(X,Es) :-
forme(X,Con,_,_,Cb,Cf),
si(Es,X),
criterefaiblesse(C),
Con<C,
nbrecas(X,N,Cb,Cf),
seuilcas(Ncas),
N>Ncas.

nbrecas(X,N,Cb,Cf) :-
N is Cb+Cf.


/*
------------------------------------------------------------------------
CONDITION SUR UNE EXPRESSION
------------------------------------------------------------------------
verifie si une expression E contient un sous arbre donné Sexp
```

```
E et Sexp sont dépouillées !!!
*/

si(E,Sexp) :-
compound(E),
E=..[Op,La,Ra],
si(La,Sexp).
si(E,Sexp) :-
compound(E),
E=..[Op,La,Ra],
si(Ra,Sexp).
/*
si(E,Sexp) :- toground(E,Eg),toground(Sexp,Gg),Eg=Gg,
        write( 'ne vous etes vous pas trompe ici'),
        write(E).
*/
si(E,Sexp) :- toground(E,Eg),toground(Sexp,Gg),Eg=Gg.



/* ------------------------------------------------------------------------ */
/*
on donne E expression numérique, et cela retourne Sexp qui a une structure
susceptible d'etre la source de l'erreur.
L est la liste de toutes les sousexpression de E qui ont une
connaisance faible
attention : ne pas oublier l'enregistrement de faux !!!
*/


/* --- on trouve dans E un sous arbre susceptible
       d'être à la source de l'ereur --- */

suiterep(Rep,E,L) :-
Rep=non,
L=[T|Q],                % si la liste est vide ca echoue
sousarbre(E,Sexp,T),
afficher(Sexp,Val),
analyserep(Sexp,Val,Rep),!.

/* --- sinon, changement de mode --- */

suiterep(Rep,E,L) :-
message(['passage en mode interactif']),solution(E).

/* --- recueil la réponse à l'erreur proposée --- */

analyserep(E,ValE,Rep) :-
globale(expression,Z),
voirevolution,
writeseqnl('Evolution',['Avez-vous ',E=ValE,' ?']),
mouinon(E,ValE,Rep).

/*
*************************************************************************

VERIFICATION PARTIELLE DE LA VALIDITE D'UNE SELECTION DS UNE EXPRESSION

*************************************************************************
Voir documentation annexe
*/

bonneselection2(E,Se) :-
effacertampon2,
```

```
write('Tampon2',E),
afficher(E,Val),
afficher(Se,Val3),
pname(Se,See),
wsearch('Tampon2',See,0,From,To),
cursor('Tampon2',From,To),
clear('Tampon2'),
cursor('Tampon2',From,From),
/* --- mettre des parentheses quand le résultat est négatif --- */
(Val3>0 -> write('Tampon2',Val3);writeseq('Tampon2',['(',Val3,')'])),
wlen('Tampon2',Long),
wsltxt('Tampon2',0,Long,T2),
pname(T22,T2),
afficher(T22,Val2),beep(10),
wlen('Tampon2',Long2),
effacertampon2,
Val=Val2.


/*
-------------------------------------------------------------------------
APPELE PAR LE DIALOGUE n° 6
-------------------------------------------------------------------------
Text est ce qui est sélectionné
Exp est la valeur de la sélection
R est la réponse proposée (atome)
Rp est la répose proposée (terme composé)
*/

/* --- on clique sur OK --- */
/*
il faut ici vérifier trois choses :
la syntaxe de la sélection doit ette correcte,
la sélection doit avoir du sens,
la valeur proposée doit etre la bonne
*/


divers(Dial,1,R,Text,Exp) :-
(cursor('Historique',Debut,Fin),not(Fin=Debut))->
(setditem(Dial,3,''),
cursor('Historique',F,T),
wsltxt('Historique',F,T,Text),
name(Text,Stext),
((syntaxe3(Dial,6,Stext),recall(vexp,Ee),
pname(Ree,Ee),pname(Tp,Text),bonneselection2(Ree,Tp))->
(miseajour(expression,Text),
% voirevolution,
% writeseqnl('Evolution',[Text]),
pname(Rp,R),
pname(Tp,Text),
afficher(Tp,Exp),
writeseqnl('Evolution',[Text]),
(Exp=:= Rp ->
(setditem(Dial,3,'Bon'),enregistrer2(Tp,1,Orig));
(setditem(Dial,3,'Faux'),enregistrer2(Tp,0,Orig),
faute(Tp,P,Rp,M,A,B,C,D),listeconfusions(faute(Tp,P,Rp,M,A,B,C,D)),listeerreur(Tp),
listevalfausse(Rp),!,fail))));
(setditem(Dial,3,'Erreur de selection !'),beep(1),fail));(setditem(Dial,3,'rien de
selectionné'),fail),!.

/* --- on clique sur Quitter --- */

divers(Dial,5,R,Text,Exp) :-
 wlen('Tampon',L),
```

```
 cursor('Tampon',0,L),
 clear('Tampon'),!,abort.

/* --- on clique sur "je donne ma langue au chat" --- */
/*
il faut ici vérifier deux choses :
la syntaxe de la selection doit etre correcte,
la sélection doit avoir du sens
*/

divers(Dial,6,R,Text,Exp) :-
(cursor('Historique',Debut,Fin),not(Fin=Debut))->
(cursor('Historique',F,T),
wsltxt('Historique',F,T,Text),
name(Text,Stext),
((syntaxe3(Dial,6,Stext)
,recall(vexp,Ee),
pname(Rp,Ee),pname(Tp,Text),bonneselection2(Rp,Tp)
) ->
(miseajour(expression,Text),
voirevolution,
writeseqnl('Evolution',[Text]),
pname(Tp,Text),
afficher(Tp,Exp),
pname(Exp,Rexp),
enregistrer2(Tp,0,Orig),
listeerreur(Tp),
listevalfausse($),
setditem(Dial,4,Rexp));
(setditem(Dial,3,'Erreur dans la selection ! '),beep(1),fail)),!);(setditem(Dial,3,'
rien de selectionné'),fail,!).


/*
-------------------------------------------------------------------------
APPELE PAR LE DIALOGUE n° 4
-------------------------------------------------------------------------
*/

/* --- on est dans la seconde "version" du dialogue --- */

syntaxerep2(D,1,Entree,X,Valeur,Rep):-
getditem('Valeur',3,_,_,_,V,_),
% ci-dessous, conditions  pour être dans la seconde version
not(V='faux'),
not(V='bon'),
not(V=''),
recall(expfausse,[N,A,B,C,De,S]),
messageerreur(N,A,B,C,De,S).

/* --- on est dans la première "version" et on clique sur OK --- */


syntaxerep2(D,1,Entree,X,Valeur,Rep) :-
miseajour(reponse,Entree),
name(Entree,String),
tst_phrase(Error,[Message,Prefix,Pos,Queue],expreponse(E),String),
Error ->
(append(Prefix,[35],J1),append(J1,Queue,Expression),
name(Exp2,Expression),setditem(D,5,Exp2),setditem(D,4,'erreur de syntaxe'),fail);
verificationresultat(D,X,Valeur,Rep).

/* --- on est dans la seconde "version" du dialogue --- */

syntaxerep2(D,6,Entree,X,Valeur,Rep):-
```

```
getditem('Valeur',3,_,_,_,V,_),
% ci-dessous, conditions  pour être dans la seconde version
not(V='faux'),
not(V='bon'),
not(V=''),
miseajour(faux,X),Rep=non,!.

syntaxerep2(D,6,Entree,X,Valeur,Rep) :- effacerevolution,abort.
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

enregistrer2(Form, Exp, Orig)
majcpt(AcptB, AcptF, NcptB, NcptF, Exp)
majprovenance(Aprov, Nprov, pers)
enregistrerfaux
effacerfauxvrai
effacerfaux
effacervrai
miseajour(Id, Var)
miseajourp(Id(Var))

************** Fin de la table des matières ************************* */

/*
*****************************************************************************

GESTION DES FAITS

*****************************************************************************

-----------------------------------------------------------------------------
MODIFICATION DU FAIT : FORME
-----------------------------------------------------------------------------
Form   : est un terme composé qui est une expression numérique
Orig   : indique l'origine de l'exercice (personnel, ordinateur ou profeseur
Exp    : caracterise le succes (1) ou l'echec (0) de l'estimation donnée par l'élève
*/

% cas absurde où on essaye d'enregistrer un nombre

enregistrer2(Form,Exp,Orig) :-
integer(Form),!.

% la stucture est rencontrée pour la première fois

enregistrer2(Form,Exp,Orig) :-
simp(Form,X),
findall(A,forme(A,_,_,_,_,_),L),not(membre(X,L)),
Prov=[0,0,0],
Liste=[0,0,0,0,0,0,0,0,0,0],
majprovenance(Prov,Nprov,Orig),
majcpt(0,0,NcptB,NcptF,Exp),
connaissance(0,Liste,Ncon,Nliste,Exp),
assert(forme(X,Ncon,Nprov,Nliste,NcptB,NcptF)),!.

% la stucture a déjà été rencontrée

enregistrer2(Form,Exp,Orig) :-
simp(Form,X),
toground(X,Xg),
forme(Xf,Con,Prov,Liste,CptB,CptF),
toground(Xf,Xfg),
Xg=Xfg,
majprovenance(Prov,Nprov,Orig),
majcpt(CptB,CptF,NcptB,NcptF,Exp),
connaissance(Con,Liste,Ncon,Nliste,Exp),
retract(forme(Xf,Con,_,Liste,CptB,CptF)),
assert(forme(Xf,Ncon,Nprov,Nliste,NcptB,NcptF)).
```

```
/*
-------------------------------------------------------------------------
  MISE A JOUR DES COMPTEURS
-------------------------------------------------------------------------
AcptB : ancienne valeur du compteur bon
AcptF : ancienne valeur du compteur faux
NcptB : nouvelle valeur du compteur bon
NcptF : nouvelle valeur du compteur faux
Exp   : résultat de l'stimation de l'élève (0 ou 1)
*/

majcpt(AcptB,AcptF,NcptB,NcptF,Exp) :-
Exp=1,
NcptB is AcptB+1,
NcptF=AcptF,!.

majcpt(AcptB,AcptF,NcptB,NcptF,Exp) :-
Exp=0,
NcptF is AcptF+1,
NcptB=AcptB.


/*
-------------------------------------------------------------------------
 MISE A JOUR DE LA PROVENANCE
-------------------------------------------------------------------------
*/

majprovenance(Aprov,Nprov,pers) :-
Aprov=[A,B,C],
Nprov=[1,B,C],!.

majprovenance(Aprov,Nprov,ordi) :-
Aprov=[A,B,C],
Nprov=[A,1,C],!.

majprovenance(Aprov,Nprov,prof) :-
Aprov=[A,B,C],
Nprov=[A,B,1].

/*
-------------------------------------------------------------------------
 ENREGISTREMENT DE LA STRUCTURE FAUSSE OU NAIT L'ERREUR
-------------------------------------------------------------------------
On met à jour l'historique et, si necessaire,
on enregistre l'expression où nait l'erreur
*/

/* ** cas où il y a erreur ** */

enregistrerfaux :-

% on rapelle l'expression de départ,l'origine de l'exerice et
% l'expresion où nait l'erreur

globale(faux,X),
globale(origine,Orig),
globale(expression,Exp),
pname(E,Exp),

% calcul de la bonne valeur

afficher(E,Ea),

% affichage adequat
```

```
writeseqnl('Historique',['Résultat incorrect.','La réponse est :',Ea]),
((recall(trouverfaute,Rep),Rep=non)->
writeseqnl('Historique',['Votre erreur nait dans l''expression  ',X]);true),
enregistrer2(X,0,Orig),!.

/* ** cas où il n'y a pas d'erreur ** */

enregistrerfaux :-
writeseqnl('Historique',['Bravo. Votre résultat est correct']).


/*
-------------------------------------------------------------------------
EFFACEMENT DE LA MEMOIRE DES FAITS BONS ET FAUX
-------------------------------------------------------------------------
*/

effacerfauxvrai :- effacerfaux,effacervrai.

effacerfaux :-
retract(globale(faux,_)),!.
effacerfaux.

effacervrai :-
retract(globale(vrai,_)),!.
effacervrai.


/*
-------------------------------------------------------------------------
MISE A JOUR DE "VARIABLES GLOBALES"
-------------------------------------------------------------------------
*/

miseajour(Id,Var) :-
retract(globale(Id,_)),assert(globale(Id,Var)),!.

miseajour(Id,Var) :- assert(globale(Id,Var)).

miseajourp(Id(Var)) :-
remember(chparametres,oui),
assert(Id(Var)),retract(Id(_)).
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

charger(X)
chargerprof(D, R)
stockerdisk(X)
inscrire(X)
sauver_source(Wlist)

************** Fin de la table des matières ************************ */

/*
*****************************************************************************

GESTION DES DONNEES PERMANENTES

*****************************************************************************
-----------------------------------------------------------------------------
CHARGEMENT DU FICHIER D'UN ELEVE
-----------------------------------------------------------------------------
ici que le fichier exite ou non tout est effacer
*/

/* --- le fichier X existe --- */

charger(X) :-
assert(faible(bidon)),
retractall(forme(_,_,_,_,_,_)),find_file(X,Vol),open(X,Vol),consult(X,Vol),close(X,Vol),

/* --- si le fichier X n'existe pas, on ne charge rien --- */

charger(X).

/*
-----------------------------------------------------------------------------
CHARGEMENT DU FICHIER DU PROFESSEUR
-----------------------------------------------------------------------------
*/

/* --- un fichier est present et il n'est pas vide --- */

chargerprof(D,R) :-  % D= début = chaine vide; R liste finale
read('prof',X),
not(X=end_of_file),
selection(D,X,S), % on ne charge que ce qui n'est pas ds la base de faits
chargerprof(S,R),!.

/* --- si pas de fichier présent --- */

chargerprof(S,S).

/*
-----------------------------------------------------------------------------
STOCKAGE DES DONNEES DANS LE FICHIER D'UN ELEVE
-----------------------------------------------------------------------------
On stocke sur disque les structures d'un eleve avec toutes les données qui
y sont attachées
*/

/* --- le fichier existe déjà --- */

stockerdisk(X) :-
```

```
findall(A,forme(A,_,_,_,_,_),L),  % ds le cas ou on commence ca echoue !
find_file(X,Vol),
create(X,Vol,'TEXT'),
map(inscrire,L),
close(X,Vol),!.

/* --- si le fichier X n'existe pas --- */

stockerdisk(X) :-
findall(A,forme(A,_,_,_,_,_),L),
create(X),
map(inscrire,L),
close(X),!.

stockerdisk(X).

inscrire(X) :-
toground(X,Xg),
globale(identifiant,W),
seek(W,2,0),
forme(Xf,B,C,D,E,F),
toground(Xf,Xfg),
Xg=Xfg,
writeseqnl(W,[forme(X,B,C,D,E,F),.]).

/*
-------------------------------------------------------------------------
ENREGISTREMENT DE LA NOUVELLE VERSION DU LOGICIEL
-------------------------------------------------------------------------
Necessaire pour sauver sur disque les nouvelles valeurs des paramètres
*/

sauver_source(Wlist) :-
recall(chparametres,Val),
Val=oui,
banner(source_save(copie_secours,Wlist),['Sauvetage des nvx paramètres']).
sauver_source(Wlist).

/* remplacer copie_secours par le nom du fichier */
```

```
/* ****************************************************************

            TABLE DES MATIERES

****************************************************************

init
menu4
<QUIT>(X)

*************** Fin de la table des matières ************************** */


/*
*******************************************************************************

INITIALISATION ET INSTALLATION DES MENUS

*******************************************************************************
*/

init :-
remember(chparametres,non),
miseajour(identifiant,bidon),
assert(forme(bidon,1,c,d,e,f)),
effacerfenetres,
creerrapport,
creerhistorique,
creerevolution,
creertampon,
creertampon2,
remember(listeexpression,[]),
menu4,
mark_item('Recherche','recherche interactive'),
mark_item('Décomposition','Par l''ordinateur'),
disable_menu('exercices'),
disable_menu('Divers'),
'Général'(nouvel_utilisateur).


menu4 :-
/*
get_menus(Lmenus),
map(kill_menu,Lmenus),
*/
install_menu('exercices',
['personnel/1','à_travailler/2','du_professeur/3','mes_bases/4']),
install_menu('Recherche',['recherche interactive','recherche auto']),
install_menu('Décomposition',['Par l''ordinateur','Par moi']),
install_menu('Divers',['rapport_général','Le_noyau','aide','configuration']),
install_menu('Général',['nouvel_utilisateur/P','quitter/Q']),
kill_menu('Fonts'),kill_menu('Edit').

init2 :-
remember(chparametres,non),
miseajour(identifiant,bidon),
assert(forme(bidon,1,c,d,e,f)),
effacerfenetres,
creerrapport,
creerhistorique,
creerevolution,
creertampon,
creertampon2,
remember(listeexpression,[]),
menu,
```

```
mark_item('Recherche','recherche interactive'),
mark_item('Décomposition','Par l''ordinateur'),
disable_menu('exercices'),
disable_menu('Divers'),
'Général'(nouvel_utilisateur).

menu :-

install_menu('exercices',
['personnel/1','à_travailler/2','du_professeur/3','mes_bases/4']),
install_menu('Recherche',['recherche interactive','recherche auto']),
install_menu('Décomposition',['Par l''ordinateur','Par moi']),
install_menu('Divers',['rapport_général','Le_noyau','aide','configuration']),
install_menu('Général',['nouvel_utilisateur/P','quitter/Q']),
kill_menu('Fonts'),kill_menu('Edit'),kill_menu('Windows'),
kill_menu('Search'),kill_menu('Eval'),disable_menu('File'),
whide('Σ Output Window')/*,disable_item('Divers','configuration'),disable_item('Divers',
aide)*/.

'<LOAD>'(X) :- init2.

'<QUIT>'(X) :- globale(identifiant,Z),stockerdisk(Z).
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

dialogueIdent(Btn, Rap)
dialogueexp(Btn, Rap)
essai(Choix, List, M)
suiteessai(Di, 3, Choix)
mondialogue(Btn, R, résultat, Val, X, Rep)
config(De1, De2, De3, De4, De5, R1, R2, R3, R4, Ve1, Ve2, Ve3, Ve4, Ve5, Vr1, Vr2, Vr3,
Vr4)
mondialogue3(Btn, R, résultat, Text, Exp)
aide
creerhistorique
creerrapport
creerevolution
creertampon
creertampon2
effacerfenetres
effacertampon
effacertampon2
garnirhistorique
voirhistorique
effacerhistorique
garnirevolution(Z)
voirevolution
effacerevolution

************** Fin de la table des matières ************************** */


/*
*****************************************************************************

LES DIALOGUES

*****************************************************************************

-----------------------------------------------------------------------------
Pour l'identification de l'utilisateur
Dialogue prioritaire
Dialogue n°1
-----------------------------------------------------------------------------
Btn est l'identifiant du dialogue
Rap renvoie un atome qui est unifié avec le nom de l'utilisateur
*/

dialogueIdent(Btn,Rap) :-
   mdialog(100,50,70,350,
           [button(10,305,20,35,ok),
           button(40,305,20,35,nul),
           text(10,10,16,150,'Identifiez vous ...'),
           edit(30,10,16,200,'nom',Rap)],Btn).


/*
-----------------------------------------------------------------------------
Pour saisir une expression et vérifier la syntaxe de l'enoncé
Dialogue prioritaire
Dialogue n°2
-----------------------------------------------------------------------------
Btn est l'identifiant du dialogue
```

```
Rap renvoie une chaine de code ASCII qui est unifié avec l'expression rentrée au
clavier
aucune verification syntaxique n'est ainsi effectuée par le système sur l'expression
entrée dans le champs
*/

dialogueexp(Btn,Rap) :-
   mdialog(200,80,70,350,
           [button(10,305,20,35,ok),
           button(40,305,20,35,nul),
           text(10,10,16,200,'Entrez votre expression'),
           edit(30,10,16,200,'expression',bytes(Rap)),
text(50,10,16,150,'')],Btn,syntaxe2(Rap)). % Rap est un atome !!

/*
-------------------------------------------------------------------------------
Zone de défilement pour l'affichage des faiblesses
Dialogue prioritaire
Dialogue n°3
-------------------------------------------------------------------------------
List est la liste des choix proposée
Choix renvoie comme atome l'expression selectionnée
M est un message
suite permet de quitter le processus
*/

essai(Choix,List,M) :- L='',dialog('Faites votre choix ',150,30,170,370,
                [button(140,10,20,60,'OK'),
                text(10,10,20,250,M),
                 button(140,210,20,60,'Cancel'),
                 menu(40,10,82,350,List,[],Choix),text(170,110,20,100,L)],
                 Btn,suiteessai(Choix)).

suiteessai(Di,3,Choix) :- abort,!.
suiteessai(Di,1,Choix) :-
(Choix=[]->message(['Faites un choix ou cancel']),!,fail).
suiteessai(Di,Bo,Choix).

/*
-------------------------------------------------------------------------------
La saisie de la valeur d'une expression et
La reponse a une proposition de regle fausse
Dialogue non prioritaire
Dialogue n°4
-------------------------------------------------------------------------------
Btn est l'identifiant du dialogue
R est unifié avec la réponse proposée par l'élève
Val est unifié avec la vraie valeur de l'expression
X est unifié avec l'énoncé de l'exercice
Val est unifié avec la valeur de l'expression X
Rep est unifié avec non ou oui si la réponse est fausse ou bonne
*/

mondialogue(Btn,R,'résultat',Val,X,Rep) :-
L=[button(35,100,20,35,ok),
text(10,10,20,100,'Votre réponse : '),
text(60,10,20,200,''),
text(85,10,20,200,''),
edit(35,10,20,70,'résultat',R),
button(35,150,20,50,'')],
dialog('Valeur',200,270,125,240,
L,Btn,syntaxerep2(R,X,Val,Rep)).


/*
```

```
-------------------------------------------------------------------------------
Dialogue pour la configuration des paramètres
Dialogue prioritaire
Dialogue n°5
-------------------------------------------------------------------------------
Di est unifié  avec les valeurs initiales des paramètres
Ri est unifié  avec les valeurs initiales des boutons radios
Vei est unifié avec les nouvelles valeurs des paramètres
Vri est unifié avec les valeurs finales des boutons radios
*/

config(De1,De2,De3,De4,De5,R1,R2,R3,R4,Ve1,Ve2,Ve3,Ve4,Ve5,Vr1,Vr2,Vr3,Vr4) :-
message('seuls le changement des 2 premiers et des 3 derniers items est implémenté !'),
dialog('Configuration',50,20,250,470,
[button(220,80,20,110;'Valider'),
button(220,280,20,110,'Ne rien changer'),
text(10,10,20,200,'Valeur de mu'),
text(35,10,20,400,'Connaissance max pour la selection des faiblesses'),
text(60,10,40,400,'Lg des listes partielles pour l''évolution de la connaissance'),
text(85,10,20,300,'Liste de départ'),
text(110,10,20,300,'nbre cas pour rech auto et rapport'),
text(135,10,20,300,'Entrer les réponses au clavier'),
radio(135,330,20,50,'Oui',R1,Vr1),
radio(135,380,20,50,'Non',R2,Vr2),
text(160,10,20,300,'Activer le filtre des exercices personnels'),
radio(160,330,20,50,'Oui',R3,Vr3),
radio(160,380,20,50,'Non',R4,Vr4),
edit(10,420,14,40,writeq(De1),read(Ve1)),
edit(35,420,14,40,writeq(De2),read(Ve2)),
edit(60,420,14,40,writeq(De3),Ve3),
edit(85,280,14,180,writeq(De4),Ve4),
edit(110,420,14,40,writeq(De5),read(Ve5))],Btn,validationgg(Ve1,Ve2,Ve5)).

/*
-------------------------------------------------------------------------------
Pour la valeur d'une  selection effectuée dans une expression
Dialogue non prioritaire
Dialogue n°6
-------------------------------------------------------------------------------
Btn est l'identifiant du dialogue
R est la valeur proposée pour la réponse de l'élève
Text est unifié avec l'expression qui est sélectionnée par l'élève
Exp est unifié avec la valeur de la sélection à partir du moment est correcte
*/

mondialogue3(Btn,R,'résultat',Text,Exp) :-
L=[button(35,100,20,35,ok),
text(10,10,20,100,'Votre réponse : '),
text(85,10,20,200,''),
edit(35,10,20,70,'résultat',R),
button(35,150,20,60,'Quitter'),
button(60,10,20,200,'Je donne ma langue au chat !')],
dialog('Valeur',200,260,125,240,
L,Btn,divers(R,Text,Exp)).

/*
-------------------------------------------------------------------------------
Pour l'aide
Dialogue non prioritaire
Dialogue n°7
-------------------------------------------------------------------------------
*/

aide :-
dialog('Les règles',180,120,150,350,
```

```
[button(120,110,20,60,'Ok'),
scrolltext(30,10,80,330,'Aide'(3,-1),'Geneva',10,0),
text(10,10,16,300,'Les formules de base')],Btn).


/*
*****************************************************************************

LES FENETRES

*****************************************************************************

-----------------------------------------------------------------------------
CREATION DES FENETRES D'AFFICHAGE
-----------------------------------------------------------------------------

*/

creerhistorique :-
is_win('Historique',disp) ->true;          %test l'existence
(wcreate('Historique',0,40,10,120,500,1),
wlock('Historique',on),                    % empeche l'écriture ds la fenetre
wfont('Historique','Courier',0,10)).       % mettre la fenetre au premier plan

creerrapport :-
is_win('rapport',disp) ->true;
(wcreate('rapport',0,50,10,280,460,1),
wfont('rapport','Courier',0,12),
wlock('rapport',on)).

creerevolution :-
is_win('Evolution',disp) ->true;
(wcreate('Evolution',0,200,10,130,250,1),
wfont('Evolution','Chicago',0,10),
wlock('Evolution',on)).

creertampon :-
is_win('Tampon',disp) ->true;
(wcreate('Tampon',0,200,10,130,250,1),
wlock('Tampon',on)).

creertampon2 :-
is_win('Tampon2',disp) ->true;
(wcreate('Tampon2',0,200,10,130,250,1),
wlock('Tampon2',on)).

/*
-----------------------------------------------------------------------------
EFFACER DES FENETRES
-----------------------------------------------------------------------------
*/

effacerfenetres :-
windows('prog',L),
map(whide,L).


effacertampon :-
 wlen('Tampon',L),
 cursor('Tampon',0,L),
 clear('Tampon').


effacertampon2 :-
```

```
wlen('Tampon2',Long3),
cursor('Tampon2',0,Long3),
clear('Tampon2').

/*
-------------------------------------------------------------------------------
FENETRE HISTORIQUE
-------------------------------------------------------------------------------
*/

garnirhistorique :-
voirhistorique,
globale(expression,A),
writenl('Historique','-------------------------------------------------------------------------------
writenl('Historique',A).

voirhistorique :-
wshow('Historique'),wfront('Historique').

effacerhistorique :-
cursor('Historique',0,-1),
clear('Historique').

/*
-------------------------------------------------------------------------------
FENETRE EVOLUTION
-------------------------------------------------------------------------------
*/

garnirevolution(Z) :-
voirevolution,
writenl('Evolution','Votre expression de départ :' ),
writenl('Evolution',Z),
writenl('Evolution',''),
writenl('Evolution','Ce que vous évaluez en ce moment :').

voirevolution :-
wshow('Evolution'),wfront('Evolution').

effacerevolution :-
cursor('Evolution',0,-1),
clear('Evolution').

/*
-------------------------------------------------------------------------------
DIALOGUE POUR LE MENU ABOUT
-------------------------------------------------------------------------------
*/
monmessage(M) :-
centred(T,L,270,350),
mdialog(T,L,270,350,[button(240,170,20,50,'Ok'),text(10,10,220,340,words(M))],Btn).
```

```
/*
**********************************************************************************

LES VALEURS DES PARAMETRES

**********************************************************************************
Le contenu de cette fenetre est mis a jour dès qu'un changement est fait

*/

coefconnaissance(0.8).
lglistepartielle(3).
listedepart([0,0,0,0,0,0,0,0,0,0]).
seuilcas(4).
criterefaiblesse(0.4).
fairetest2(oui).
reponseentree(oui).
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

rapportg
ecrirefamille(X)
attacheverdict(X)
ecrireverdict(X)
composer(A, B)
verdict(Con, L, Cb, Cf, Message)

************** Fin de la table des matières ************************ */



/*
*****************************************************************************

RAPPORT GLOBAL SUR UN ELEVE

*****************************************************************************
-----------------------------------------------------------------------------
APPEL PRINCIPAL
-----------------------------------------------------------------------------
*/


rapportg :-
cursor('rapport',0,-1),
clear('rapport'),

/* première partie du rapport*/

findall(A,attacheverdict(A),L),
keysort(L,Ls,-1),
writeseqnl('rapport',['Sur l''évolution de la maîtrise des calculs']),
writeseqnl('rapport',['']),
map(ecrireverdict,Ls),

/* seconde partie du rapport */

writeseqnl('rapport',['']),
writeseqnl('rapport',['']),
writeseqnl('rapport',['Sur les familles de calcul : ']),
writeseqnl('rapport',['']),

noyau(T),
(not(T=[]) ->
map(ecrirefamille,T);write('rapport','Rien à dire pour le moment')),
wshow('rapport'),
wfront('rapport').

ecrirefamille(X) :-
writeseqnl('rapport',['Des difficultés avec ',X]),!.
/*
-----------------------------------------------------------------------------
DIAGNOSTIQUE
-----------------------------------------------------------------------------
*/


attacheverdict(X) :-
```

```
forme(F,Con,_,L,Cb,Cf),
garnir(F,Fg),
pname(Fg,Fga),
verdict(Con,L,Cb,Cf,M),
X=M-Fga.

/*
-------------------------------------------------------------------------
ECRIRE LE DIAGNOSTIQUE
-------------------------------------------------------------------------
*/

ecrireverdict(X) :-
X=M-Fga,
composer(Fga,M).


/*
-------------------------------------------------------------------------
FORMATTAGE DE L'ECRITURE DU DIAGNOSTIQUE
-------------------------------------------------------------------------
 attention, il est impératif d'imprimer en courier pour l'espacement entre
les caractères et l'alignement des remarques
*/

composer(A,B) :-
fw('rapport',[c(-35)],[A,B]),nl('rapport'). % voir p127

/*
-------------------------------------------------------------------------
POSER LE DIAGNOSTIQUE
-------------------------------------------------------------------------
Voir documentation annexe
Ct est le seuil fixé par paramètrage pour avoir une "bonne" connaissance
Con est la valeur de la connaissance
L est la liste des résultat
Cb est le nombre de fois que l'expression a été bien évaluée
Cf est le nombre de fois que l'expression a été mal évaluée
*/

/* --- la connaissance est supérieure au seuil fixé --- */

verdict(Con,L,Cb,Cf,Message) :-
criterefaiblesse(Ct),
Con>Ct,
Message='bonne connaissance',!.

/* --- sinon, l'expression n'a pas été rencontrée
suffisament de fois pour juger --- */

verdict(Con,L,Cb,Cf,Message) :-
lglistepartielle(Lg),
Cs is Cb+Cf,
Ce is Lg+2,
Cs<Ce,
Message='donnees insuffisantes',!.

/* --- sinon, les derniers résultats sont meilleurs --- */

verdict(Con,L,Cb,Cf,Message) :-
lglistepartielle(Lg),
connaissancelp(L,C),
C>Con+0.5,
Message='en progres',!.
```

```
/* --- sinon, les derniers résultats sont plus mauvais --- */

verdict(Con,L,Cb,Cf,Message) :-
lglistepartielle(Lg),
connaissancelp(L,C),
C<Con-0.5,
Message='en recul',!.

/* --- sinon, on ne sait rien dire --- */

verdict(Con,L,Cb,Cf,Message) :-
Message='evolution incertaine'.
```

```
/* ************************************************************

            TABLE DES MATIERES

************************************************************

syntaxe2(D, Btn, S)
syntaxe3(D, Btn, S)
tst_phrase(Error, [Message, Prefix, Pos, Queue], G, I)
tst_syntaxe(Message, Queue, Queue)
exp(E)
add_exp(E)
add_exp_q(D, E)
add_exp_q(D, D, X, X)
mul_exp(E)
mul_exp_q(D, E)
mul_exp_q(D, D, X, X)
int_exp(E)
digits(S)
tdigits([S| T])
ndigits([S| T])
ndigits([], X, X)
digit(C)
add_op(+)
mul_op(*)
l_par
r_par
w
w(X, X)
testexpresion(I)
expreponse(E)
add_exp2(E)
mul_exp2(E)
mul_exp_q2(D, E)
mul_exp_q2(D, D, X, X)
int_exp2(I)
digits2(S)
tdigits2([S| T])
ndigits2([S| T])
ndigits2([], X, X)
digit2(C)
mul_op2(/)

*************** Fin de la table des matières ************************** */

/*
-------------------------------------------------------------------------
APPELE PAR LE DIALOGUE n° 2
-------------------------------------------------------------------------
*/

syntaxe2(D,Btn,S) :-
tst_phrase(Error,[Message,Prefix,Pos,Queue],exp(E),S),
(Error ->(setditem(D,5,'erreur de syntaxe !!'),append(Prefix,[35],J1),append(J1,Queue,
Expression),
name(Exp2,Expression),setditem(D,4,Exp2),fail);
(transfo(R,S),(integer(R)->(message(['Pas de nombre entier !']),fail);pname(Rr,R),
miseajour(expression,R),not(fractionentiere2(Rr)),true))).



/*
-------------------------------------------------------------------------
UTILISÉ PAR LA DECOMPOSITION MANUELLE
-------------------------------------------------------------------------
```

```
Ce prédicat est employé dans la fonction où l'élève selectionne
la partie qu'il veut réduire.
Le dialogue numéro 4 a ses champs éventuellement mis à jour
*/


syntaxe3(D,Btn,S) :-
tst_phrase(Error,[Message,Prefix,Pos,Queue],exp(E),S),
(Error ->(setditem(D,3,'erreur de selection !'),append(Prefix,[35],J1),
append(J1,Queue,Expression),
name(Exp2,Expression),fail);
(transfo(R,S),(integer(R)->(message(['Pas de nombre entier ! ']),fail);pname(Rr,R),
miseajour(expression,R),not(fractionentiere2(Rr)),true))).

/*
nécessaire car pour le logiciel name(R,""11-3"") donne pour R un terme composé
et avec name(R,""11+5-6") R est un atom !!!!

transfo est déjà defini ds une autre fenêtre

transfo(A,S) :-
name(R,S),atom(R),A=R,!.

transfo(A,S) :-
name(R,S),pname(R,A).
*/

/*
*****************************************************************

Les prédicats de cette fenêtre ont pour objectifs de reconnaître la validité
syntaxique des expressions arithmétiques entrées par l'utilisateur.

En cas d'erreur syntaxique, l'expression est refusée et un message indiquant autant
que faire se peut la nature et la position de l'erreur est retourné.
***************************************************************** /*

/*================================================================

ANALYSE DES ERREURS

================================================================*/

/* ----------------------------------------------------------------

Prédicat d'analyse de phrase avec message d'erreur :

tst_phrase(Error,[Message,Préfixe,Pos,Queue],exp(E),Input)
    - Error est instancié à true ou false selon que l'expression
        est reconnue ou non;
    - Message est le libellé du message d'erreur éventuel;
    - Préfixe est la partie de l'input reconnue (précédant
        l'erreur reconnue);
    - Pos est la position du dernier caractère reconnu dans
        le string d'entrée;
    - Queue est la partie non reconnue dans le string d'entrée;
    - exp(E) est une expression référençant le non terminal
        principal d'une grammaire DCG;
    - Input est la chaîne d'entrée à analyser par ladite grammaire.

   Ce prédicat réussit toujours.
   En cas d'erreur, E est instancié à une expression dans laquelle la partie non
reconnue de l'input est remplacée par une variable non-instanciée.
----------------------------------------------------------------*/
```

```
tst_phrase(Error,[Message,Prefix,Pos,Queue],G,I):-
 remember(err_message,[]),
 phrase(G,I)
 ->Error=false
  ;(Error=true,
    recall(err_message,[Message,Queue]),
    append(Prefix,Queue,I),
    length(Prefix,Pos)).

/* ------------------------------------------------------------------

Prédicat permettant de signaler une erreur dans la description de la grammaire :

tst_syntaxe(Message)
    - Message est une valeur qui sera retournée si l'analyse échoue à partir de ce
point.

    Ce prédicat doit être utilisé sans {} dans la description de la grammaire.
------------------------------------------------------------------*/

tst_syntaxe(Message,Queue,Queue). % first ignore,
tst_syntaxe(Message,Queue,[]):-   % on backtracking, signal error
 recall(err_message,M),
 (M=[]                            % is it allready a signaled error?
 ->(remember(err_message,
             [Message,Queue]),fail)  % not yet, remember
  ;(M=[X,Queue]                      % yes, at same input position?
    ->(remember(err_message,
                [Message,Queue]),fail)  % yes, recall most general
     ;fail                              % no, ignore
    )).

/*==================================================================

SYNTAX DESCRIPTION

==================================================================*/


exp(E)-->
 add_exp(E).

/*------------------------------------------------------------------
 Les non-terminaux de la forme x_q indiquent la "queue" d'un non terminal x. L'intérêt
est d'éviter la répétition de la reconnaissance d'une même structure comme dans les
règles :

exp(E)-->term(E).
exp(Op(L,R))-->term(L),add_op(Op),exp(R).
------------------------------------------------------------------*/

add_exp(E)-->
 [45],mul_exp(D),!,  % 45 est le code ASCII du signe "-" !!!
 add_exp_q(D,E).

add_exp(E)-->
 mul_exp(D),
 add_exp_q(D,E).



add_exp_q(D,E)-->
 add_op(Op),!,
 tst_syntaxe('mul exp expected'),
 mul_exp(N),
```

```
 add_exp_q(Op(D,N),E).
add_exp_q(D,D,X,X). % add expression queue is empty

mul_exp(E)-->
 int_exp(D),
 mul_exp_q(D,E).


mul_exp_q(D,E)-->
 mul_op(Op),!,
 tst_syntaxe('int exp expected'),
 int_exp(N),
 mul_exp_q(Op(D,N),E).
mul_exp_q(D,D,X,X). % mul expression queue is empty

int_exp(E)-->
 l_par,!,
 tst_syntaxe('sub-exp expected'),
 exp(E),
 r_par,!.
int_exp(I)-->
 tst_syntaxe('number expected'),
 digits(S),
 {name(I,S)}.

/*-----------------------------------------------------------------
On utilise un CUT pour forcer l'acceptation d'un digit s'il est présent. On utilise un
prédicat intermédiaire pour forcer digits à contenir au moins un chiffre. On utilise
un autre prédicat intermédiaire pour empêcher les espaces inclus.
-----------------------------------------------------------------
*/
digits(S)-->w,tdigits(S).

tdigits([S|T])-->digit(S),!,ndigits(T).

ndigits([S|T])-->digit(S),!,ndigits(T).
ndigits([],X,X).



digit(C)-->[C],
            {C=<"9",            %Plus simple que le code ASCII
            "0"=<C}.            % Les {} sont nécessaires pour
indiquer qu'il s'agit
                                % d'une condition

add_op(+)-->w,"+".          % n'y a-t-il pas plus simple pour
skipper les blancs ?
add_op(-)-->w,"-".

mul_op(*)-->w,"*".
mul_op(/)-->w,"/".

l_par-->w,"(".

r_par-->w,")".

w-->" ",!,w.                        % on force les espaces à être pris
w(X,X). % When no more spaces, accept.

/*================================================================

TESTS

================================================================*/
```

```
testexpresion(I):-
 wfront(user),
 tst_phrase(Error,[Message,Prefix,Pos,Queue],exp(E),I),
 (Error
  ->(nl(),
     fw(user,[b(0)],[Prefix,Queue]),nl(),
     tab(Pos),
     writenl('^'),
     writenl(Message))
   ;(V is E,fw(user,[q(0),' = ',q(0)],[E,V]),nl())).



/*
:-testexpression("6+3*4")

/*============================================================== */


/*
le prédicat qui suit est employé exclusivement pour tester la syntaxe de la valeur
proposée par un élève pour une expression donnée. Cette valeur ne peut être qu'un
entier où un nombre fractionnaire; Il faut donc que la syntaxe rejette toute autre
forme.
*/

expreponse(E)-->
 add_exp2(E).


add_exp2(E)-->
 mul_exp2(D).



mul_exp2(E)-->
 int_exp2(D),
 mul_exp_q2(D,E).

mul_exp_q2(D,E)-->
 mul_op2(Op),!,
 tst_syntaxe('int exp expected'),
 int_exp2(N).
mul_exp_q2(D,D,X,X). % mul expression queue is empty


int_exp2(I)-->
 tst_syntaxe('number expected'),
 digits2(S),
 {name(I,S)}.

/*-----------------------------------------------------------------
On utilise un CUT pour forcer l'acceptation d'un digit s'il est présent. On utilise un
prédicat intermédiaire pour forcer digits à contenir au moins un chiffre. On utilise
un autre prédicat intermédiaire pour empêcher les espaces inclus.
-----------------------------------------------------------------
*/
digits2(S)-->w,tdigits2(S).

tdigits2([S|T])-->digit2(S),!,tst_syntaxe('pas de lettre'),ndigits2(T).
tdigits2([S|T])-->[45],digit2(S),!,tst_syntaxe('pas de lettre'),ndigits2(T).
% 45 est le code ASCII du signe "-" !
ndigits2([S|T])-->digit2(S),!,ndigits2(T).
ndigits2([],X,X).
```

```
digit2(C)-->[C],{C=<"9", "0"=<C}.

mul_op2(/)-->w,"/".

/*
w-->" ",!,w.                          % on force les espaces à être pris
w(X,X). % When no more spaces, accept.
*/
```

```
/* ****************************************************************

            TABLE DES MATIERES

****************************************************************

fraction(D/G, [D, G])
plus(A+B, G)
moins(A-B, G)
fois(A*B, G)
divi(A/B, G)
eval(X, R)
pgcd(I, 0, I)
reduire(X, R)
connaissance(OldC, Cl, Nc, [Exp| Rest], Exp)
remove_last([Elem, Q], [Elem])
connaissancelp(L, C)
listepartiel(L, Lg, N)
lp(L, D, Cptdepart, Lg, S)
sousarbre(E, Sexp, Mod)
configuration
valeurB1(B1, R1, R2)
valeurB2(B2, R3, R4)
validationg(D, B, Ve1, Ve2, Ve5)
validation(Ve1)
validation2(Ve2)
validation3(Ve5)
compilation
lesmiseajours(Ve1, Ve2, Ve3, Ve4, Ve5, Vr1, Vr2, Vr3, Vr4)
random(N)
afficher(X, E)
ecrire(R, E)
identifier
test(X, D, D)
testenonce(Form)
fractionentiere(X)
fractionentiere2(X)
souvenir
pertinence(X, Lg)
solution(X)
killdouble(L, M, D)
double([], A, A, D, D)
killdouble(L, M)
double([], A, A)
killtriple(D, R, T)
noyau(Tt)
leNoyau(L)
voir(List, M)
voirnoyau
modeevolution(X, E)
selection(D, X, S)
solutiong(X)
validationgg(D, B, Ve1, Ve2, Ve5)
transfo(A, S)
transfo2(A)
membre(X, [Y| __1])
effacerblanc
effacer(__1, [], [])

************** Fin de la table des matières ************************* */


/*
*********************************************************************
```

DES CALCULS

```
*************************************************************************
-------------------------------------------------------------------------
CALCUL DE LA VALEUR D'UNE EXPRESSION
-------------------------------------------------------------------------
X est un terme composé
Le résultat de chaque opération est systématiquement transformé en fraction
*/


fraction(D / G,[D,G]) :- integer(D), integer(G),!.
%-----un nombre entier est mis sous forme fractionnaire-----
fraction(X,R) :- integer(X),R=[X,1].


plus(A + B,G) :-
A=[L,M],
B=[P,Q],
N is L*Q+M*P,
Z is Q*M,
G=[N,Z].



moins(A - B,G) :-
A=[L,M],
B=[P,Q],
N is L*Q-M*P,
Z is Q*M,
G=[N,Z].

fois(A * B,G) :-
A=[L,M],
B=[P,Q],
N is L*P,
Z is Q*M,
G=[N,Z].

divi(A/B,G) :-
A=[L,M],
B=[P,Q],
N is L*Q,
Z is P*M,
G=[N,Z],Z=0,message(['denominateur nul']),fail,!.

divi(A/B,G) :-
A=[L,M],
B=[P,Q],
N is L*Q,
Z is P*M,
G=[N,Z],not(Z=0).


eval(X,R) :- fraction(X,R).
eval(X + Y,R) :- eval(X,T),eval(Y,U),plus(T+U,R),!.
eval(X - Y,R) :- eval(X,T),eval(Y,U),moins(T-U,R),!.
eval(X * Y,R) :- eval(X,T),eval(Y,U),fois(T*U,R),!.
eval(X / Y,R) :- eval(X,T),eval(Y,U),divi(T/U,R),!.
eval(-X,R) :- eval(X,T),T = [A,B],R=[-A,B].

/*
-------------------------------------------------------------------------
```

```
CALCUL DU PGCD DE DEUX NOMBRES
-------------------------------------------------------------------------
Algorithme d'Euclide
K est le pgcd de I et J
*/

pgcd(I,0,I) :- !.
pgcd(I,J,K) :- I>0,Z is I, mod(Z,J,R), pgcd(J,R,K),!.
pgcd(I,J,K) :- Z is -1*I,mod(Z,J,R),pgcd(J,R,K).

/*
-------------------------------------------------------------------------
REDUCTION D'UNE FRACTION
-------------------------------------------------------------------------
X est une liste de deux éléments
R est une liste de deux éléments : [numératur,dénominateur]
R est "réduit" par rapport à X
*/

reduire(X,R) :-
 X=[A,B],
 pgcd(A,B,S),
 N is A/S,
 D is B/S,
 R=[N,D].


/*
-------------------------------------------------------------------------
CALCUL DE LA CONNAISSANCE
-------------------------------------------------------------------------
Voir documentation annexe
*/

connaissance(OldC,Cl,Nc,[Exp|Rest],Exp):-
 coefconnaissance(Mu),
 map(++,Cl,0,S), % p 23
 Nc is S*0.1*Mu +(1-Mu)*OldC,
 remove_last(Cl,Rest).

remove_last([Elem,Q],[Elem]):-!.
remove_last([Elem|Q],[Elem|Rest]):-
 remove_last(Q,Rest).



/*
-------------------------------------------------------------------------
CALCUL DE LA CONNAISSANCE sur une liste partielle issue de L
-------------------------------------------------------------------------
Voir documentation annexe
*/

connaissancelp(L,C) :-
lglistepartielle(Lg),
listepartiel(L,Lg,N),map(++,N,0,R),C is R/3.


/*
-------------------------------------------------------------------------
CONSTUCTION DE LA LISTE PARTIELLE DE LG ELEMENTS
-------------------------------------------------------------------------
*/

listepartiel(L,Lg,N) :- D=[],Cptdepart=0,lp(L,D,Cptdepart,Lg,N).
```

```
/*
ci dessous, tout est donné sauf N qui est la liste partielle de longeur Lg
*/

lp(L,D,Cptdepart,Lg,S) :-
L=[T|Q],
append(D,[T],N),
Lg1 is Cptdepart+1,
Lg1≤Lg,
lp(Q,N,Lg1,Lg,S),!.

lp(Q,N,Cptdepart,Cptfin,N) .



/*
-------------------------------------------------------------------------
RECHERCHE SI UNE EXPRESSION CONTIENT UNE STRUCTURE DONNÉES
-------------------------------------------------------------------------
 ici Sexp est composée de variables et E est numérique
exemple : sousarbre(5+6*9,sexp,_*_) renvoie 6*9
*/

sousarbre(E,Sexp,Mod) :-
compound(E),
E=..[Op,La,Ra],
sousarbre(La,Sexp,Mod).

sousarbre(E,Sexp,Mod) :-
compound(E),
E=..[Op,La,Ra],
sousarbre(Ra,Sexp,Mod).

sousarbre(E,Sexp,Mod) :-
simp(E,Es),
toground(Es,Eg),
toground(Mod,Mods),
Eg=Mods,
Sexp=E.



/*
*************************************************************************

DES PARAMETRES

*************************************************************************
-------------------------------------------------------------------------
APPEL DU MENU
-------------------------------------------------------------------------
*/

configuration :-
% appel des paramètres
coefconnaissance(De1),
criterefaiblesse(De2),
lglistepartielle(De3),
listedepart(De4),
seuilcas(De5),
fairetest2(B2),
valeurB2(B2,R3,R4),
reponseentree(B1),
valeurB1(B1,R1,R2),
```

```
% appel du dialogue
config(De1,De2,De3,De4,De5,R1,R2,R3,R4,Ve1,Ve2,Ve3,Ve4,Ve5,Vr1,Vr2,Vr3,Vr4),
banner(lesmiseajours(Ve1,Ve2,Ve3,Ve4,Ve5,Vr1,Vr2,Vr3,Vr4),['Mise a jour des paramètres']
compilation.          % appel au compilateur.



valeurB1(B1,R1,R2) :-
B1=oui ->(R1=on,R2=off);(R1=off,R2=on).

valeurB2(B2,R3,R4) :-
B2=oui ->(R3=on,R4=off);(R3=off,R4=on).

validationg(D,B,Ve1,Ve2,Ve5) :-
validation(Ve1),
validation2(Ve2),
validation3(Ve5).

validation(Ve1) :-
number(Ve1),Ve1>0,Ve1<1,!.
validation(Ve1) :-
message(['Un nombre compris entre 0 et 1 ']),fail.

validation2(Ve2) :-
number(Ve2),Ve2>0,Ve2<1,!.
validation2(Ve2) :-
message(['Un nombre compris entre 0 et 1 ']),fail.

validation3(Ve5) :-
integer(Ve5).

validation3(Ve5) :-
message(['Un nombre entier']),fail.

/*
-------------------------------------------------------------------------
COMPILATION
-------------------------------------------------------------------------
si on ne le fait pas, ce sont toujours les anciennes valeurs qui, interviennent dans
le calcul. En plus, il faut sauver le nouveau texte.

*/

compilation :-
wsyntax('Paramètres',Syntaxe,Mode),
cursor('Paramètres',0,0),   % pour que toute la fenetre soit compilee
banner(Syntaxe('Paramètres',Mode),['Compilation des paramètres ...']).

/*
-------------------------------------------------------------------------
LES MISES A JOUR DES PARAMETRES
-------------------------------------------------------------------------
Les différentes valeurs sont successivement mises à jour
*/

lesmiseajours(Ve1,Ve2,Ve3,Ve4,Ve5,Vr1,Vr2,Vr3,Vr4) :-
miseajourp(coefconnaissance(Ve1)),
miseajourp(criterefaiblesse(Ve2)),
miseajourp(seuilcas(Ve5)),
(Vr1=on ->miseajourp(reponseentree(oui));miseajourp(reponseentree(non))),
(Vr3=on ->miseajourp(fairetest2(oui));miseajourp(fairetest2(non))).


/*
*************************************************************************
```

```
UTILITAIRES

*************************************************************************
-------------------------------------------------------------------------
ENGENDRE UN NOMBRE ENTIER ALEATOIRE NON NUL
-------------------------------------------------------------------------
générer des nombres aléatoires supérieurs à 0; irland(A,N) génére des nombres
entre o at N. Il suffit donc d'ajouter 1 pour éviter les 0!
*/

random(N) :- irand(19,Nr),N is Nr+1.

/*
-------------------------------------------------------------------------
FORMATTE L'AFFICHAGE DE LA VALEUR D'UNE EXPRESSION
-------------------------------------------------------------------------
regle l'affichage suivant que le resultat est un nombre entier ou une fraction
*/

afficher(X,E) :-
  eval(X,A),
  reduire(A,R),
  ecrire(R,E).


ecrire(R,E) :-  R=[A,1],E=A,!.
ecrire(R,E) :-  R=[A,B],not(B=1),E=A/B.



/*
-------------------------------------------------------------------------
IDENTIFICATION DE L'UTILISATEUR
-------------------------------------------------------------------------
Identifie et mémorise l'utilisateur
*/

identifier :-
dialogueIdent(Choix,X),
Choix=1,
miseajour(identifiant,X),
effacerhistorique,
effacerevolution,
banner(charger(X),['Chargement en cours du fichier',X]).

/*
-------------------------------------------------------------------------
TESTS POUR NE CHARGER QUE LES FAITS QUI NE SONT PAS PRESENR DANS LA BASE DE DONNEES
-------------------------------------------------------------------------

*/

/*

appel ds chargement du fichier professeur

*/
test(X,D,D) :-
simp(X,Xs),
toground(Xs,Xss),
forme(Z,_,_,_,_,_),
toground(Z,Zs),
Xss=Zs,!.
```

```
test(X,D,S) :- append(D,[X],S).

/*
-------------------------------------------------------------------------
TEST SI UNE STRUCTURE EST DEJA BIEN CONNUE OU NON
-------------------------------------------------------------------------
*/

testenonce(Form) :-
fairetest2(oui),
simp(Form,X),
toground(X,Xg),
criterefaiblesse(Vcritique),
forme(Xf,C,_,_,_,_),
toground(Xf,Xfg),
Xfg=Xg,
C>Vcritique,!,
warning(['vous connaissez déjà bien.~M Poursuivre malgre tout ?']),!.

testenonce(Form).



/*
-------------------------------------------------------------------------
TEST SUR UNE EXPRESSION
-------------------------------------------------------------------------
*/



fractionentiere(X) :-
X=A/B,integer(A),integer(B).

/*
ON NE PEUT SELECTIONNER UNE FRACTION ENTIERE
*/

fractionentiere2(X) :-
X=A/B,integer(A),integer(B), message(['Pas de fraction entière !!']).


/*
-------------------------------------------------------------------------
LISTE DES STRUCTURES DÉJÀ VUES AU COURS D'UNE SESSION
-------------------------------------------------------------------------
Ce but réussit toujours

Es est la structure de E
L est la liste des structures déjà vues
L1 est la liste des srtuctures de L semblables à Es
Long est le nombre d'éléments de L1
Nl est la liste L augmentée de Es
*/

/* --- il y a des structures déjà vues semblables à Es --- */

souvenir :-
recall(listeexpression,L),
globale(expression,E), % E est un atome
pname(Exp,E),
simp(Exp,Es),
toground(Es,Esg),
findall(A,(on(A,L),toground(A,Ag),Ag=Esg),L1),
length(L1,Long),
```

```
not(Long=0),
Lg is Long+1,
pertinence(Es,Lg),
L=[R],
Nl=[Es,R],
remember(listeexpression,Nl),!.

/* --- il n'y a pas de structures déjà vues semblables à Es --- */
/* --- car L est vide --- */

souvenir :-
recall(listeexpression,L),
globale(expression,E), % E est un atome
pname(Exp,E),
simp(Exp,Es),
toground(Es,Esg),
L=[],
Nl=[Esg],
remember(listeexpression,Nl),!.

/* --- il n'y a pas de structures déjà vues semblables à Es --- */
/* --- et L n'est pas vide --- */


souvenir :-
recall(listeexpression,L),
globale(expression,E), % E est un atome
pname(Exp,E),
simp(Exp,Es),
toground(Es,Esg),
Nl=[Esg|L],
remember(listeexpression,Nl).



/*
-------------------------------------------------------------------------
CONSULTATION DE LA CONNAISSANCE D'UNE STRUCTURE DÉJÀ VUE
-------------------------------------------------------------------------
Ce but réussit toujours
*/

pertinence(X,Lg) :-
criterefaiblesse(Critere),
findall(Ag,(forme(A,Con,_,_,_,_),Critere>Con,toground(A,Ag)),L),
toground(X,Xg),on(Xg,L),!.

/*
si le predicat qui précede échoue, cela ne veut pas dire que l'expression est
nécessairement bien connue. Il se pourrait que l'expression de départ ne soit pas dans
la base de données quand on décompose en sélectionnant.Avant de lancer le message, on
s'assure que l'expression est connue
*/


pertinence(X,Lg) :-
findall(Ag,(forme(A,Con,_,_,_,_),not(Critere>Con),toground(A,Ag)),L),
toground(X,Xg),on(Xg,L),
writeseqnl('Historique',['Avertisement : cette structure est essayée pour la ',Lg,'
fois durant cette session et vous la connaissez bien ']),
warning(['vous connaissez déjà bien.~M Poursuivre malgre tout ?']),!.


/*
-------------------------------------------------------------------------
```

```
LANCE L'ANALYSE D'UNE EXPRESSION
-------------------------------------------------------------------------
lance la recherche de la localisation de l'erreur
X  est un terme composé
Z  est un terme composé
Zp est un atome
*/


solution(X) :-
globale(expression,Z),
pname(Z,Zp),
garnirevolution(Z),
explorer2(X).


/*
-------------------------------------------------------------
ELIMINATION DES DOUBLES
-------------------------------------------------------------
L est la liste entrée
M est la liste d'entrée moins les éléments de D
D est la liste des éléments présents au moins deux fois
*/

killdouble(L,M,D) :- double(L,[],M,[],D).

double([],A,A,D,D):-!.
double([T|Q],A,L,D,G) :- on(T,A), double(Q,A,L,[T|D],G),!.
double([T|Q],A,L,D,G) :- double(Q,[T|A],L,D,G).

/* --- idem sans D --- */

killdouble(L,M) :- double(L,[],M).
/* le second argument sert à accumuler  CM p146 */
double([],A,A):-!.
double([T|Q],A,L) :- on(T,A), double(Q,A,L),!.
double([T|Q],A,L) :- double(Q,[T|A],L).
/* ------------------------ */

/*
-------------------------------------------------------------
ELIMINATION DES TRIPLES
-------------------------------------------------------------
D est la liste de départ
R est la liste de départ moins les éléments présents moins de trois fois
T est la liste des éléments présents au moins trois fois
*/
/* Test la liste des éléments au moins en triple; D liste de départ */
killtriple(D,R,T) :-
killdouble(D,L,Double),
killdouble(Double,R,Tp),
killdouble(Tp,T).

/*
-----------------------------------------------------------------
NOYAUX ** LISTE DES STRUCTURES PRESENTE AU MOINS TROIS FOIS ** NOYAUX
-----------------------------------------------------------------
*/

noyau(Tt) :-
listeformes(L,Lsd),
map(garnir,L,Lg),
map(decapiter,Lg,Ldecapiter),% on ne peut décapiter que des structures garnies
map(garnirlettre,Ldecapiter,Ll),
```

```
% killtriple ne s'applique qu'a des expressions littérales

killtriple(Ll,R,T),
map(pname,T,Tt).

leNoyau(L) :-
noyau(T),
not(T=[]),
L=T,!.

leNoyau(L) :-
L=['Rien pour le moment'].


voir(List,M) :- dialog('Le coeur des difficultés ',150,30,120,370,
                [button(90,90,20,60,'OK'),
                text(10,10,20,250,M),
                menu(30,10,60,350,List,[],Choix)],
                 Btn).

voirnoyau :-
leNoyau(L),
voir(L,'Le noyau').



/*
*****************************************************************************

CHOIX PARAMETRES

*****************************************************************************
*/

modeevolution(X,E) :-
reponseentree(oui),
writenl('Evolution',X),!.

modeevolution(X,E) :-
writeseqnl('Evolution',['Avez-vous ',X=E,'?']).

/* ---------------------------------------------------------------------------- */

% appel danss chargement du fichier profeseur

selection(D,X,S) :-
test(X,D,S),!.  % module "Utilitaires"

selection(D,X,S) :- append(D,[X],S).

/* ---------------------------------------------------------------------------- */

solutiong(X) :-
marked_item('Recherche','recherche interactive'),
solution(X),!.
solutiong(X) :-
auto(X).


/*
-------------------------------------------------------------------------
APPELE PAR LE DIALOGUE n° 5
-------------------------------------------------------------------------
```

```
*/

validationgg(D,B,Ve1,Ve2,Ve5) :-
banner(validationg(D,B,Ve1,Ve2,Ve5),['Contrôle de validation en cours ...']).


/*
-------------------------------------------------------------------------
UTILITAIRES DIVERS
-------------------------------------------------------------------------
*/


/*
nécessaire car pour le logiciel name(R,""11-3"") donne pour R un terme composé
et avec name(R,""11+5-6") R est un atom !!!!
Ce prédicat transforme toujours A en un atome qd S est donné
*/

transfo(A,S) :-
name(R,S),atom(R),A=R,!.

transfo(A,S) :-
name(R,S),pname(R,A).

transfo2(A) :-
integer(A),!.
transfo2(A) :-
A = -C, integer(C).

/* -------------------------------------------------------------------------
teste si une structure X fait partie d'une liste de structures
*/

membre(X,[Y|_]) :- not(Y=bidon),toground(X,F),toground(Y,F),!.
membre(X,[_|Y]) :- membre(X,Y).


/*
Permet d'avoir un affichage correct dans la fenetre historique, car le logiciel ajoute,
dans certaines circonstances, des blancs dans la fenetre Tampon par où transitent les
expressions séléctionnées
*/

effacerblanc :-
wlen('Tampon',L),
wsltxt('Tampon',0,L,A),
name(A,La),
effacer(32,La,Lb),
name(B,Lb),
effacertampon,
write('Tampon',B).

effacer(_,[],[]).
effacer(X,[X|L],M) :- !,effacer(X,L,M).
effacer(X,[Y|L1],[Y|L2]) :- effacer(X,L1,L2).
```