MASTER EN SCIENCES INFORMATIQUES

Outils logiciels d'aide à l'apprentissage du concept de nombre

Goffaux, Alain

Award date:
1996

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix

Namur

Institut d'Informatique

Outils logiciels d'aide à l'apprentissage du concept de nombre.

Promoteur

Claude CHERTON

Mémoire présenté par

Alain GOFFAUX

en vue de l'obtention du titre de

licencié et maître en informatique.

Année Académique 1995-1996

Facultés Universitaires Notre-Dame de la Paix

Namur

Institut d'Informatique

Outils logiciels d'aide à l'apprentissage du concept de nombre.

Promoteur

Claude CHERTON

Mémoire présenté par

Alain GOFFAUX

en vue de l'obtention du titre de

licencié et maître en informatique.

Année Académique 1995-1996

Je me permets tout d'abord d'adresser mes remerciements à mon promoteur M. Cherton pour sa disponibilité et ses conseils tout au long de ce mémoire.

Ma reconnaissance va ensuite à M. Doneux, Inspecteur Cantonal de l'Enseignement subventionné de la Communauté Française, pour sa précieuse collaboration lors de la réalisation de la partie pédagogique de cet ouvrage.

Un grand merci enfin à la famille et aux amis qui m'ont aidé à l'élaboration de ce travail.

Partie 1 : GENERALITES.

1. Introduction.

En 4ème année d'humanités, l'élève apprend à résoudre une équation du second degré.

Que lui dit-on ?

a) Si $a \neq 0$, on a

$$ax^2 + bx + c = 0 \Leftrightarrow x^2 + \frac{b}{a}x + \frac{c}{a} = 0,$$

$$\Leftrightarrow \left(x + \frac{b}{2a}\right)^2 - \frac{b^2}{4a^2} + \frac{c}{a} = 0,$$

$$\Leftrightarrow \left(x + \frac{b}{2a}\right)^2 = \frac{b^2 - 4ac}{4a^2}.$$

On pose

$$\boxed{\rho = b^2 - 4ac} \quad \text{(réalisant)}$$

Si	alors l'équation $ax^2 + bx + c = 0$ $(a \neq 0)$ a, dans $\mathbb{R}$,
$\rho < 0$	aucune solution
$\rho = 0$	une solution $x_1 = -\frac{b}{2a}$
$\rho > 0$	deux solutions $x_1 = \frac{-b + \sqrt{\rho}}{2a}$ et $x_2 = \frac{-b - \sqrt{\rho}}{2a}$

car, si $\rho \geq 0$, cette équation équivaut à

$$x + \frac{b}{2a} = \pm\sqrt{\frac{\rho}{4a^2}} = \pm\frac{\sqrt{\rho}}{2a}.$$

b) Noter que, si $ac < 0$, on a $\rho > 0$, donc l'équation a deux solutions.

c) *Règle simplifiée*

Si $b = 2b'$, on peut simplifier les formules : on a

$$\rho = (2b')^2 - 4ac = 4(b'^2 - ac) = 4\rho',$$

si on pose

$$\rho' = b'^2 - ac.$$

On remarque que

$$\rho \left\{\begin{matrix} > \\ = \\ < \end{matrix}\right\} 0 \Leftrightarrow \rho' \left\{\begin{matrix} > \\ = \\ < \end{matrix}\right\} 0$$

et, si $\rho' \geq 0$, on a

$$\frac{-b \pm \sqrt{\rho}}{2a} = \frac{-2b' \pm 2\sqrt{\rho'}}{2a} = \frac{-b' \pm \sqrt{\rho'}}{a}.$$

Tous les facteurs 2 ont donc disparu des formules !

On lui propose alors des exercices qu'il résout sans trop de difficultés en appliquant la formule qu'il a apprise, mémorisée.

Quelques années plus tard, l'élève retombe sur une telle équation. Malheureusement, par manque d'application de la formule, il est dans l'impossibilité de s'en rappeler. Il a oublié le truc, il se retrouve dans une impasse. Pourquoi ?
Parce qu'il appliquait, par le passé, la formule comme un artifice lui permettant de résoudre une équation du second degré sans savoir exactement ce qu'elle signifiait, sans savoir ce qu'il faisait et pourquoi il le faisait. Il est fort à parier que s'il ne s'était pas seulement contenté de mémoriser la formule mais également de comprendre (et retenir) le raisonnement permettant d'aboutir à cette formule, quelques instants auraient été suffisants pour qu'il la reconstruise et puisse résoudre l'équation face à laquelle il se retrouve.

Cet exemple montre bien toute l'importance de la compréhension dans l'apprentissage. Si le savoir-faire est important, la compréhension l'est davantage.

2. Contexte et existant.

Une brève description du but de ce mémoire.

Ce travail se situe au croisement de deux grands domaines occupant une position très importante dans notre vie à tous : l'éducation et l'informatique. Le premier est à la base de notre culture, de notre manière de vivre, de notre personnalité. Le second prend une part de plus en plus importante tant au niveau de la vie professionnelle qu'au niveau privé. Le but de ce travail est de fournir des outils informatiques, logiciels destinés à l'apprentissage du concept de conservation ou encore de l'invariance numérique.

L'enseignement évolue.

L'enseignement n'est pas un domaine immuable. Les réformes ne sont pas choses courantes mais elles existent. D'ailleurs, notre époque est sans doute la plus explicite: réformes, manifestations , aménagements, ...

S'il s'agit ici de réformes concernant plutôt les institutions que la pédagogie, les réformes touchant plus spécifiquement cette dernière existent. Prenons un exemple concret à savoir l'enseignement en France.

Quand l'école publique est née en France, à la fin du siècle dernier, le programme pour les enfants de 5 à 6 ans était l'étude des "Quatre opérations sur des nombres à deux chiffres". Cette apprentissage constituait l'essentiel du programme d'arithmétique depuis la maternelle jusqu'au cours moyen (4-5ème). Le but du caractère répétitif du programme était de bien graver dans la mémoire de l'enfant ce qui était (re)vu.

En évitant de poser des additions avec retenues, il était relativement facile d'apprendre à des enfants de 5 - 6 ans le calcul de sommes telles que 21 + 63. Il suffisait de les entraîner à positionner ces nombres verticalement en respectant l'alignement en colonnes et il suffisait à l'enfant d'effectuer les calculs simples que sont 2 et 6 et 1 et 3.

Bien entendu, les enfants des classes maternelles ne peuvent pas comprendre pourquoi ce mode de calcul est valide. D'autre part, la somme de 21 et 63 est déjà un grand nombre auquel un élève ne peut attribuer un sens. Le but pédagogique était que l'enfant prenne très tôt de bonnes habitudes.

Apparaît alors une première réforme. En 45, un nouveau programme est appliqué. Selon ce programme, en grande section (3ème maternelle), les quatre opérations ne doivent plus faire l'objet que de petits exercices de calcul mental et les exercices de calcul écrit doivent être accompagnés de dessins correspondants. Toutefois, en 60, il était encore fréquent que les enfants des classes maternelles mettent les additions en colonnes.

En 70, les réformateurs (réforme des mathématiques modernes) mettent fin aux additions en colonnes en maternelle mais également aux premières connaissances du calcul mental. L'enfant ne devait plus calculer du tout. On eut alors recours à des activités dites pré numériques tels que la mise en ordre de bâtonnets ou bien l'emboîtement de cubes.

Enfin, en 86, de nouvelles instructions officielles françaises indiquent qu'à l'école maternelle l'enfant 'apprend et récite la comptine numérique'.

Ce petit historique de l'enseignement français en maternelle montre bien que l'enseignement n'est pas un domaine immuable mais évolutif.

La tendance actuelle en Belgique.

Chez nous, à notre époque et du point de vue pédagogique, la tendance est à l'individualisation et plus précisément à la différenciation des apprentissages. Cette nouvelle pédagogie est proposée dans le décret de l'école de la réussite (Mars 95). L'enseignant ne travaille plus uniquement avec toute une classe sur un même contenu mais il commence à travailler avec un enfant ou avec un groupe d'enfants en fonction de leurs points faibles et de leurs besoins. Nous montrerons en quoi l'informatique peut être utile dans ce contexte (Voir: Pourquoi utiliser l'ordinateur ?).

L'informatique et l'enseignement.

Si le micro-ordinateur a envahi les entreprises, il commence peu à peu à s'implanter dans les foyers mais également dans les écoles. Dans ce contexte, son utilisation se résume généralement à deux points :

- La Bureautique (Secrétariat, gestion du personnel, communication, ...)
- L'enseignement de l'informatique.

Toutefois, ces dernières années, est apparue, petit à petit, l'utilisation de la machine comme support didactique. Ces derniers temps, la tendance s'est encore accrue. Pour preuve, nous pouvons voir de temps en temps un spot publicitaire vantant la qualité de tel ou tel logiciel didactique.

Quels types de programmes liés à l'éducation, à l'apprentissage trouve-t-on aujourd'hui sur le marché ?

Propres à l'enseignement.

Les tutoriels.

C'est un enseignement programmé. Des leçons sont proposées à l'enfant avec contrôle de la compréhension de l'enfant grâce à la présentation d'exercices

pré définis. Le passage d'une leçon à l'autre est soit linéaire soit laissé à la guise de l'utilisateur.

Les logiciels de 'drill'.

Il s'agit de logiciels présentant à l'enfant des exercices d'un certain type, voire de plusieurs types, avec des valeurs pré définies ou alors aléatoires. Leurs fonctions sont d'interroger l'utilisateur, de corriger les réponses, de lui indiquer le résultat de ses réponses et éventuellement de lui indiquer les réponses correctes en cas d'erreur.

Les logiciels exerciseurs correcteurs explicatifs.

Il s'agit de logiciels de 'drill' auxquels on a ajouté une dimension pédagogique à savoir le rôle explicatif. Le logiciel ne se contente pas de dire si la réponse donnée est correcte ou pas mais il tente d'expliquer, de faire comprendre à l'utilisateur ce qu'il est entrain de faire ou la raison pour laquelle il s'est trompé.

Non typiques à l'enseignement.

Les logiciels de gestion électronique de documents.

Ce sont des logiciels qui sont à l'intersection de 3 types de logiciels :

. le traitement de textes.
. le système de gestion de bases de données
. le logiciel hypertexte (voir hypermédia).

Comme exemple, on pourrait citer une encyclopédie qu'il serait possible d'actualiser.

Les simulateurs.

Leurs objectifs :

- Aider à comprendre un phénomène, une loi physique, ...
- Evaluer un temps, une durée (ex : temps d'accès à un élément d'une base de données en fonction de son volume).
- Entraîner (ex : entraîner le personnel à réagir à un problème technique dans une centrale nucléaire).
- Evaluer plusieurs hypothèses pour un choix optimum (ex : certains logiciels d'aide à la décision ont recours à de telles évaluations).

Les logiciels composés.

Un logiciel qui serait à la fois tutoriel et exerciseur correcteur explicatif.

Les logiciels de développement.

Ce sont des logiciels permettant aux enseignants de développer leur propres leçons ou exercices, dans le sens 'liberté quant à l'évolution de ceux-ci'.

3. Une brève description des objectifs de l'enseignement.

Toute définition d'un objectif doit normalement répondre aux 5 questions suivantes :

- Qui ?
- Quoi ?
- Quand ?
- Comment ?
- Pourquoi ?

Qui sont les personnes visées ?

L'enseignement, à priori, concerne toutes les personnes.

Quoi ? Que veut-on faire ?

Apprendre à l'individu à s'exprimer (parole et écriture), permettre l'accès à la connaissance (lecture), apprendre à vivre avec d'autres individus (politesse, comportement en société, tolérance, ...), ...

Pour ce qui est de l'enseignement fondamental, le CEF (Conseil de l'Education et de la Formation) a défini les 3 finalités suivantes :

- l'école doit amener les jeunes à être des citoyens responsables en favorisant l'apprentissage des responsabilités sociales, par l'exercice de la démocratie à l'école;

- elle doit promouvoir l'épanouissement individuel de chaque jeune en développant la confiance en soi, l'autonomie, la tolérance et la solidarité envers les autres;

- elle doit conduire les jeunes à participer à la vie économique, en les aidant à devenir acteurs de leur propre formation et constructeurs de leur propre savoir.

Quand ?

Tout au long de la jeunesse et même par après si la nécessité s'en fait sentir (spécialisation, recyclage, ...).

Comment ?

En instruisant les individus c'est-à-dire en diffusant une grande partie des connaissances et en les aidant à les comprendre et à les appliquer.

Pourquoi ?

Pour faciliter l'intégration de l'individu à la société et pour le bien-être de l'humanité.

4. Objectifs de ce mémoire.

Qui va-t-on aider ?

L'instituteur de 3ème maternelle - 1ère primaire.
L'enfant.

A quoi va-t-on les aider ?

L'instituteur : On va l'aider - lors du développement d'applications pédagogiques fonctionnant sous windows et étant étroitement liées à la notion de conservation.

- à combler son absence.

L'enfant : On va l'aider à acquérir la notion de conservation du nombre.

Quand va-t-on les aider ?

L'instituteur : Tout au long de sa carrière professionnelle et lorsqu'il désire développer une nouvelle application.

L'enfant : Au moment où il a atteint l'âge d'apprendre la notion de conservation du nombre.

Comment va-t-on les aider ?

L'instituteur : - En lui proposant une bibliothèque d'objets programmés qu'il pourra utiliser lors du développement de ses applications liées à la notion de conservation.

Nous ne prétendons pas permettre à l'instituteur non initié à l'informatique de développer seul une application. La bibliothèque peut être la plus évoluée et la plus simple possible, généralement, l'enseignant se fera assister d'un informaticien et ceci est d'autant plus vrai s'il s'agit d'une bibliothèque d'objets destinée aux applications Windows. Seul l'instituteur passionné par l'informatique sera peut-être apte à développer seul des applications. Toutefois, nous pensons que les objets définis permettrons à l'instituteur de créer sur papier une ébauche de l'application ce qui facilitera le travail de l'informaticien.

- En lui donnant un logiciel développé à partir de cette bibliothèque et traitant de la notion de conservation du nombre.

Vous trouverez dans la section 'Pourquoi utiliser l'ordinateur ?' les rôles que joue ou que peut jouer ce logiciel.

L'enfant : En lui proposant ce même logiciel l'aidant à acquérir la notion de conservation du nombre. La manière de faire comprendre à l'enfant ce qu'il fait, la

manière de lui faire comprendre cette notion de conservation recourt essentiellement à la perception visuelle. La raison est que l'enfant amené à découvrir le concept de conservation ne maîtrise pas encore très bien la langue de Voltaire. Bien entendu, les manipulations d'objets concrets par les enfants garderont leurs prérogatives. Il ne faudrait pas croire que la seule exploitation de logiciels pourrait se suffire à elle-même dans la lente conquête de la conservation du nombre.

Pourquoi ?

L'instituteur : - Car certains enseignants sont passionnés par leur métier et veulent profiter des nouvelles technologies pour améliorer leur travail et aider au maximum les enfants lors de leur apprentissage.

- Car nous nous dirigeons vers un enseignement où les classes sont de plus en plus nombreuses et deviennent par conséquent des lieux où se côtoient des gens dont les capacités sont très différentes. Et pendant qu'un enseignant travaille avec un groupe, il faut occuper les autres enfants de manière idéale c'est-à-dire en prenant en compte leurs besoins. Développer des applications pédagogiques sur lesquelles les enfants pourront s'exercer est une solution.

L'enfant : Car l'enseignement maternel et primaire constituent la base des apprentissages ultérieurs et qu'il est donc essentiel que l'enfant comprenne parfaitement les concepts abordés.

5. Pourquoi utiliser l'ordinateur ?

L'objectif initial de ce mémoire était l'élaboration d'outils informatiques permettant d'aider l'enfant lors de l'apprentissage du concept "nombre" et des 4 opérations fondamentales en arithmétique que sont l'addition, la soustraction, la multiplication et la division.

Il est évident que ce domaine est bien trop large que pour pouvoir être entièrement couvert par un seul mémoire.

Toutefois, quelle que soit la matière à laquelle on s'attaque, une question apparaît :

Pourquoi utiliser l'ordinateur alors que cet apprentissage peut se faire à l'aide de différents objets (dans notre cas : réglettes, livres d'exercices, ...) ?

Si l'ordinateur ne fait que retranscrire sur écran ce qui existe déjà, nous ne voyons pas l'utilité, l'intérêt de son usage. Il faut donc que l'utilisation de l'ordinateur apporte un plus à ce qui existe déjà.

Voyons ce que l'ordinateur va nous apporter via un didacticiel.

L'ordinateur en tant que correcteur explicatif.

Lorsqu'un enfant fait des exercices,

- soit il s'agit d'exercices que l'enseignant lui a demandé de réaliser

- soit il s'agit d'une initiative personelle de l'enfant afin de s'améliorer, de s'exercer ou encore afin de se divertir.

Dans le premier cas, une fois que l'enfant a terminé, il remet son travail à l'enseignant, celui-ci le corrige, et lui accorde une note avec éventuellement des félicitations. L'enfant peut donc voir où il s'est trompé, mais personne ni aucune note ne lui indique pourquoi il s'est trompé, qu'elle a été son erreur. De plus, la plupart du temps, la bonne réponse ne lui est même pas indiquée.

Dans le second cas, c'est encore pire. Si les parents, le grand frère ou la grande soeur ne sont pas derrière lui, n'ont pas de temps à lui accorder, l'enfant se sera exercé mais n'aura même pas connaissance de ses erreurs.

L'ordinateur pourrait jouer ici un rôle très important, le rôle de l'enseignant ou des parents absents, **le rôle de correcteur**. Toutefois, si on limite la capacité, l'intérêt de l'ordinateur à ce simple rôle, son apport n'est que très minime et on peut se demander si cela vaut la peine de dépenser de l'argent à l'achat d'ordinateurs.
Par contre, cela devient intéressant lorsque l'ordinateur ne se contente pas uniquement de corriger mais également de justifier pourquoi il y a erreur, d'expliquer

l'erreur. Ce rôle explicatif de l'ordinateur est un apport non négligeable pour ne pas dire essentiel.

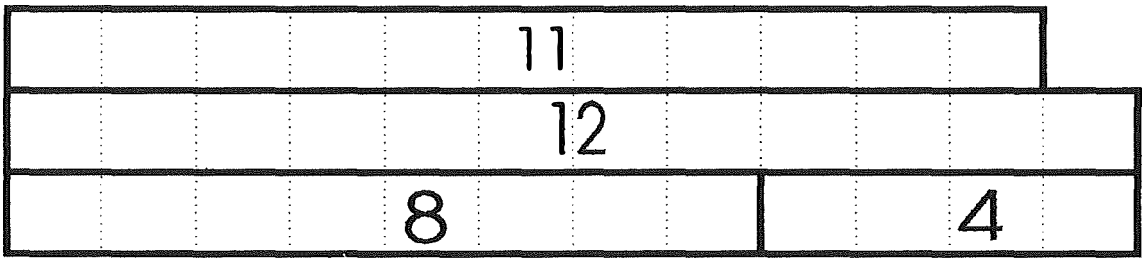
Un exemple pour mieux comprendre.

Supposons que le problème auquel on s'attaque soit l'addition et qu'on demande à l'enfant de remplir l'égalité lacunaire suivante :

8 + 4 = ...

Supposons que l'enfant réponde 11

On pourrait lui indiquer son erreur de la manière suivante afin qu'il puisse voir où il s'est trompé, quelle a été son erreur.



L'ordinateur en tant que <u>témoin de progression.</u>

Généralement, l'enseignant n'a pas assez de temps pour évaluer l'évolution de chaque enfant dans l'apprentissage. Ceci est d'autant plus vrai s'il doit s'occuper de plusieurs classes à la fois. Seuls, les enfants en très grosses difficultés seront repérés.

L'ordinateur peut jouer ce rôle ou du moins permettre à l'enseignant de repérer en quelques instants les enfants à la traîne. Si le didacticiel comprend un mécanisme permettant à l'enseignant de visualiser le niveau de chaque élève, le professeur pourra facilement détecter les enfants en difficultés. Il pourra alors les prendre en charge spécifiquement afin de les aider.

L'ordinateur en tant que <u>suppléant.</u>

La tendance actuelle dans l'enseignement primaire est à la différenciation des apprentissages et à la continuité.

De quoi s'agit-il ?

La tendance actuelle n'est plus seulement de travailler globalement avec toute une classe sur un sujet mais de travailler avec les enfants par groupe en fonction de leurs points faibles et de leurs besoins.

Par exemple, si l'enseignant voit qu'un enfant n'éprouve pas de difficulté en orthographe mais est en échec en arithmétique, il mettra davantage l'accent sur l'arithmétique, travaillera davantage cette matière avec lui (et éventuellement d'autres ayant les mêmes difficultés).

Il est évident que pendant que l'enseignant travaille une certaine matière avec un groupe d'élèves, les autres doivent s'occuper. Et c'est ici que l'ordinateur peut jouer le rôle de suppléant. Pendant que l'instituteur travaille avec un groupe, les autres enfants pourraient s'occuper sur ordinateur. Il faut remarquer que ce rôle de suppléant est fonction des applications exécutables. Au plus les applications couvriront des domaines et seront intéressantes, meilleur sera ce rôle de suppléant.

6. Objectifs d'un didacticiel.

Nous présentons ci-dessous les objectifs généraux des didacticiels liés à l'enseignement. Selon le type de didacticiel (tutoriel, exerciseur, outil de développement, ...), certains objectifs risquent de faire défaut.

Qui ? Quelles sont les personnes visées ?

L'enfant.
L'enseignant.

Quoi ? Que va-t-on faire ?

Aider l'enfant lors de l'apprentissage d'un concept (ex : le concept de nombre) ou d'une action (ex: taper à la machine).

Aider l'enseignant à enseigner, à évaluer l'apprentissage de l'enfant, à développer, généralement avec l'aide d'un informaticien, ses propres logiciels ou à développer ses propres leçons ou types d'exercices.

Quand ?

L'enfant : Lorsqu'il en a besoin.

L'enseignant : Tout au long de sa vie professionnelle.

Comment ?

L'enfant : En lui proposant des leçons ou exercices lui permettant de comprendre le concept tout en tenant compte de son niveau dans l'apprentissage ou, dans le cas d'une action, de comprendre ce qu'il fait (Savoir-faire <> Compréhension).

L'enseignant : En lui fournissant du déjà prêt (ex : une leçon programmée), en lui proposant un suivi pour chaque enfant, un langage de programmation de très haut niveau ou une bibliothèque d'objets lui facilitant le développement de logiciel ou en lui proposant un logiciel lui permettant de créer ses propres leçons ou applications.

Pourquoi ?

L'enfant : Car l'éducation est à la base de toute notre existence et influence fortement celle-ci. Il semble cependant que l'enseignement amène de plus en plus les enfants à acquérir un savoir-faire alors que son but est de faire comprendre à l'enfant un concept ou une opération. L'introduction montre bien toute l'importance de la compréhension par rapport au savoir-faire. De plus, il apparaît une certaine déresponsabilité des parents envers leurs enfants. Ceux-ci sont de plus en plus livrés

à eux-mêmes. Quels sont les parents qui font encore vivre des expériences d' empilements, de dénombrements, ... à leur(s) enfant(s) ? Pis, quels sont les parents qui vérifient si leurs enfants ont bien fait leurs devoirs et quels sont leurs résultats ?

L'enseignant : Car nous nous dirigeons vers un enseignement où les classes sont de plus en plus nombreuses et deviennent par conséquent des lieux où se côtoient des personnes dont les capacités sont très différentes. Et pendant qu'un enseignant travaille avec un groupe, il faut occuper les autres enfants de manière idéale c'est-à-dire en prenant en compte leurs besoins.

Partie 2 : ANALYSE PEDAGOGIQUE.

7. Support pédagogique.

- La notion de collection.
- La notion de collection-témoin
- La notion de conservation.
- La notion de quantité.
- Autres apprentissages liés à la conservation.

Remarque : Pour les trois premières notions, le mot 'quantité' est utilisé au sens usuel.

A) La notion de collection.

Une collection est une réunion d'objets possédant généralement les mêmes caractéristiques.

B) La notion de collection-témoin.

Le nombre n'est pas le seul moyen dont on dispose pour garder en mémoire ou communiquer une quantité.

Pour représenter une quantité, une pratique autrefois courante consistait à construire une collection-témoin par correspondance : s'il s'agissait de représenter la quantité d'un troupeau de moutons, le berger constituait un tas de cailloux, un caillou représentant un mouton. Le tas de cailloux ainsi formé constituait une collection-témoin qui permettait de garder en mémoire le nombre de moutons que comptait le troupeau. Quand le berger revenait des pâturages, il faisait correspondre un caillou pour chaque mouton ce qui lui permettait de savoir si la quantité de moutons était conservée ou, dans le cas contraire, comment elle avait évolué.

Le principe de base du procédé est la correspondance terme à terme. La quantité est représentée par l'ensemble des éléments mis en correspondance terme à terme. Ces éléments constituent une collection-témoin. Une collection-témoin est donc une représentation analogique des quantités.

Le terme analogique provient du fait que lorsqu'une quantité est représentée par une collection-témoin, elle se présente sous une forme très similaire à celle sous laquelle elle est perçue : Quatre moutons sont représentés par quatre cailloux. C'est ainsi qu'on peut dire de la représentation d'une quantité par une collection-témoin qu'elle est une représentation analogique.

Vous trouverez dans la section 'Au-delà de Piaget' une autre définition de la collection-témoin. Nous ne la mentionnons pas ici, car elle fait référence à un autre concept non encore défini.

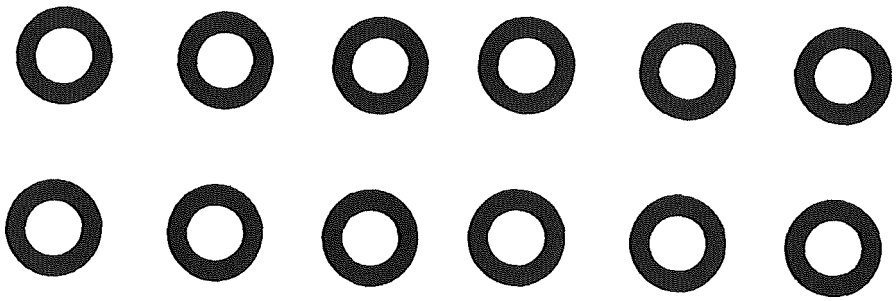
C) La notion de conservation (ou encore de l'invariance).

1) Une première approche.

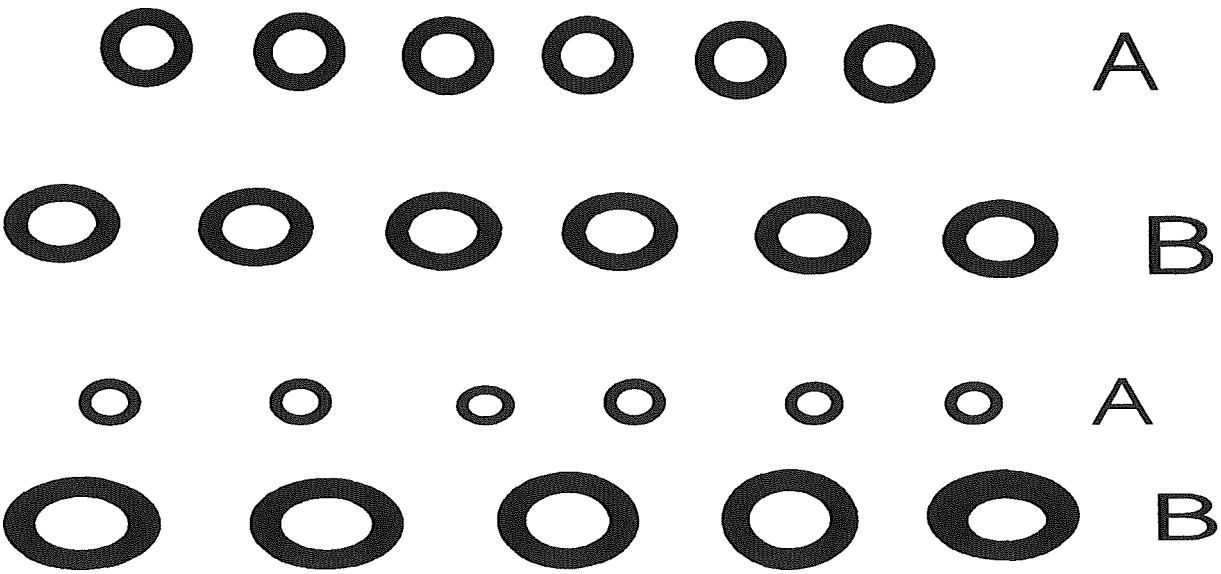
De quoi s'agit - il ?

Le problème de la conservation a été exploré par Piaget.
Il s'agit que l'enfant conserve le nombre quand changent la nature des objets, leur couleur, leur forme, leur grandeur, leur situation dans l'espace.

Si vous demandez, dans le cas de la figure ci-dessous, à un enfant de 5 ans, quel est le tas contenant le plus d'éléments, il vous répondra qu'il y a égalité entre les deux.

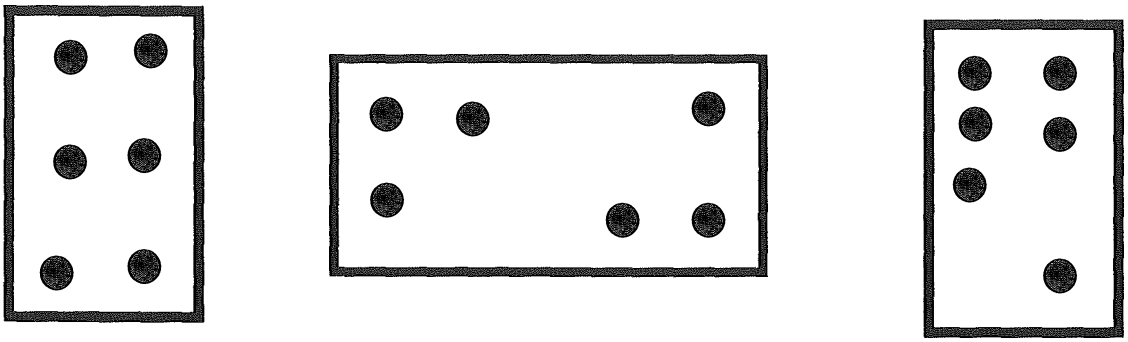


Si, par contre, vous présentez une des configurations suivantes, il faudra attendre que l'enfant ait environ 6 ans pour que le "autant que" soit bien établi. Avant cet âge, l'enfant répondra généralement qu'il y en a plus dans le tas B parce que celui-ci occupe plus de place, est "plus gros".



Le même type de problème va naître lorsque la disposition spatiale est modifiée.

Exemple :



Ce problème apparaît même si la transformation s'effectue sous les yeux des enfants. En fait, les enfants ne conservent pas le nombre lorsqu'on passe d'une disposition à l'autre. Ils vont recourir chaque fois au dénombrement pour arriver à dire qu'il y en a 6.

Remarque.
Il pourrait sembler étrange de déjà parler de dénombrement à ce niveau. Mais un enfant sortant de maternelle est déjà apte à dénombrer un certain nombre d'objets pour autant que ceux-ci ne soient pas plus de 6.

Comment expliquer ces difficultés ?

En fait, les enfants perçoivent la situation de départ ainsi que la situation d'arrivée mais ne retiennent pas le film des transformations que subissent les choses, film qui leur permettrait de conserver l'égalité des cardinaux au nom du principe : rien n'a été ajouté, rien n'a été retiré, donc, c'est autant. Pour ces enfants le réel fluctue sans cesse. Des permanences ne sont pas assurées et en particulier, celle qui assure la constance du nombre.

Piaget distingue les types de conservation suivants :

- Conservation d'ensembles numériques
- Conservation des quantités continues - discontinues
- Conservation des longueurs
- Conservation substance - poids - volume

la conservation du nombre étant généralement considérée comme la réunion des deux premiers types.

2) Approche approfondie.

Les explications qui vont suivre ont été tirées de l'oeuvre '*La genèse du nombre chez l'enfant*' de Piaget et Szeminska.

Qu'est-ce que la conservation (l'invariance) ?

La conservation est une idée qui est partout et toujours postulée par l'esprit à titre de condition nécessaire de toute intelligence mathématique.

En effet, un ensemble ou une collection n'est concevable que si leur valeur totale demeure inchangée quels que soient les changements introduits dans les rapports des éléments. Un nombre n'est également intelligible que dans la mesure où il demeure identique à lui-même quelle que soit la disposition des unités dont il est composé. C'est ce qu'on appelle l'invariance du nombre.

Du point de vue psychologique, le besoin de conservation constitue donc une sorte d'a priori fonctionnel de la pensée, c'est-à-dire qu'au fur et à mesure de son développement, ce besoin s'impose nécessairement.

Comment la conservation se construit-elle chez l'enfant?

Ce mémoire n'ayant pas un objectif purement pédagogique, il nous semble qu'une explication simple mais compréhensible de l'évolution de la construction de la conservation chez l'enfant prévaut sur une explication lourde et fastidieuse.

Quel que soit le type de conservation, la construction de la notion de conservation passe par les trois stades suivants :

- Stade1. : L'absence de conservation.
- Stade 2. : Les réponses intermédiaires.
- Stade 3. : La conservation nécessaire.

Afin d'expliquer de la manière la plus claire qui soit ces stades et le passage de l'un à l'autre, nous nous baserons sur un cas concret à savoir la construction de la conservation de quantités discontinues.

De quoi s'agit-il ?

On présente à l'enfant deux récipients de mêmes dimensions (R1 et R2) contenant le même nombre de perles, puis on verse le contenu de R2 dans un récipient de forme différente (RD) ou dans deux récipients plus petits et semblables (RP1 et RP2) pour demander à l'enfant si la quantité transvasée de R2 en RD ou en (RP1+RP2) est restée égale à celle de R1. Au besoin, on peut ensuite verser les perles de RP1 dans deux récipients égaux entre eux et plus petits encore (RPP1 et RPP2) puis le cas échéant, verser les perles de RP2 dans deux autres récipients (RPP3 et RPP4) identiques à (RPP1 et RPP2), on pose alors une nouvelle question d'égalité, ...

STADE 1. : L'absence de conservation.

Durant le premier stade, il n'y a pas conservation des collections de perles. Non seulement l'enfant croit à des changements de quantité globale lorsqu'on transvase une collection quelconque d'un récipient dans un autre de forme différente mais encore il croit que le collier confectionné avec les perles ne sera pas de même longueur dans les deux cas.

Explications.

En fait, les quantités sont d'abord évaluées simplement en fonction de rapports perceptifs (différences de longueur, de hauteur) non coordonnés entre eux et c'est cette incohérence initiale qui explique l'absence de tout critère de conservation.

L'enfant est persuadé que la quantité croît ou diminue en fonction de la forme ou du nombre de récipients. Tout changement perçu est considéré comme entraînant une modification de la quantité totale de perles. Tout se passe comme si l'enfant ignorait la notion de quantité totale ou multidimensionnelle et ne pouvait jamais raisonner que sur une seule relation (largeur, longueur, hauteur) à la fois sans les coordonner aux autres.

Dès le contact perceptif avec l'objet, il convient donc de chercher le principe de différenciation entre la quantité et la qualité. Toute perception attribue en effet des qualités à des objets mais les enfants ne peuvent appréhender ces qualités sans les mettre par le fait même en relation les unes avec les autres. Ces relations ne sauraient être que de deux sortes :

. les rapports symétriques qui expriment les ressemblances et
. les rapports asymétriques qui expriment les différences.

Or les différences de quantités impliquent le plus et le moins et marquent ainsi le début de la quantification. Sous sa forme élémentaire, la quantité est donc donnée en même temps que la qualité : elle est constituée par les rapports asymétriques qui relient nécessairement entre elles les qualités quelles qu'elles soient. On parle, à ce stade, de "quantité brute" ou de "qualité brute".

Pour mieux faire sentir l'égalité, non pas globale, mais élément à élément, de deux collections, on peut faire mettre par l'enfant une perle dans un récipient donné toutes les fois que l'on en met une dans le récipient parallèle. Or cette correspondance biunivoque et réciproque ne suffit pas non-plus à assurer la conservation. L'enfant comprend bien que les deux collections correspondantes sont égales lorsqu'elles sont situées dans deux récipients de même forme mais il suffit de verser R1 dans un récipient de forme différente RD pour que la collection contenue dans R1 ne soit plus considérée comme égale à celle de RD.

Ceci s'explique par le fait que la quantification, à ce stade, est si peu poussée que la correspondance n'entre pas en conflit avec les apparences contraires et se subordonne d'emblée à la perception spatiale.

STADE 2. : Les réponses intermédiaires.

Ce stade est caractérisé par des réponses ou solutions intermédiaires situées à mi-chemin entre la quantité brute (ou qualité brute) sans invariance et la quantification proprement dite. Il y a conflit entre la correspondance (construction des collections par correspondance bi-univoque et réciproque) et l'apparence contraire (différence de niveau ou de largeur).

Il s'agit d'un véritable conflit c'est-à-dire que les facteurs de conservation ne se soumettent pas sans plus aux facteurs d'altération mais qu'on assiste à une lutte dont les péripéties sont de plus en plus constructives. Ensuite, et à cause de cela, les rapports perceptifs se coordonnent en relations et s'intègrent ainsi en un système susceptible de justifier la conservation tout en rendant compte des variations coexistantes.

Toutefois, à ce stade, il n'y a conservation pour l'enfant que lors d'un changement peu important.

A cause même des hésitations de l'enfant à admettre la conservation en cas de changements de forme, l'enfant est conduit à dissocier les évaluations fondées sur la seule perception des rapports de hauteur ou de largeur et celles qui résultent de la représentation des longueurs des colliers. Il y a conservation lorsque l'enfant pense à l'alignement des perles et non-conservation lorsqu'il pense à l'une ou l'autre des dimensions de la forme globale.

Il y a un conflit systématique et sans issue entre un facteur d'égalité et de conservation et un facteur de différence. Lorsque l'enfant regarde les collections de perles il croit à la non-équivalence et lorsqu'il se rappelle la correspondance qui les a constituées il croit à nouveau à cette équivalence.

STADE 3. : Conservation et coordination quantifiante.

A ce stade l'enfant n'a plus à réfléchir pour s'assurer de la conservation des quantités totales : il est certain à priori.

Il y a coordination des relations et ces dernières sont concentrées en un acte unique au lieu de se constituer pas à pas.

Les réponses données (qui sont correctes à ce stade) émanent d'une multiplication logique des relations en jeu de hauteur et de largeur. Pour lever la contradiction entre la correspondance bi-univoque et réciproque entre les éléments de la collection, source d'équivalence, et les changements apparents, le sujet suppose d'emblée que ceux-ci forment un tout.

Mais dans le cas où l'équivalence des contenus des récipients n'a pas été établie préalablement, une telle opération de multiplication des relations[1] ne suffit pas à

[1] La multiplication des relations est l'opération qui consiste à coordonner des différences (ex : différences de hauteur et de largeur).

constituer la notion de quantité constante ou d'égalité de deux quantités. Mais une fois en possession de cette multiplication des relations, l'enfant fait l'hypothèse que les différences peuvent être égalées. Les différences perçues sont mesurées et, à défaut de données numériques, elles sont mesurées les unes aux autres, toute augmentation de largeur étant égalée ou comparée à la diminution concomitante de hauteur ou l'inverse.

Ces égalisations de différences, les proportions ou les partitions numériques se constituent en fonction des opérations inverses dont l'enfant acquiert le maniement par le fait même de rendre 'opératoires' les transformations jusque là conçues à de simples rapports perceptifs. C'est cette réversibilité propre à toute opération logique et mathématique qui permet de concevoir égalisations et décompositions.

Résumé :

A tous les niveaux, l'enfant est naturellement porté à croire que deux collections qui se correspondent terme à terme sont équivalentes.

Stade 1. : Pas de conflit entre conservation et perception car les rapports perceptifs l'emportent d'emblée sur l'équivalence.

Stade 2. : Les facteurs en présence sont de forces égales.

Stade3. : L'équivalence prime d'emblée les rapports perceptifs.

Le passage progressif du premier stade au troisième s'explique de la manière suivante :

1) Les simples rapports perceptifs deviennent (commencement au deuxième stade) de véritables relations que les enfants coordonnent et engendrent ainsi des systèmes de graduation ou de quantité intensive. Un rapport perceptif ne constitue pas comme tel une relation. Le critère de l'existence psychologique des relations est la possibilité de leur composition (addition et multiplication)[2]. Mais même si la coordination des relations par l'enfant était effectuée intégralement par les enfants du deuxième stade, elle ne suffirait point à les conduire à la conservation des quantités totales sauf si la hauteur et la largeur étaient simplement permutées. Un récipient de perles qui augmente en hauteur et diminue en largeur peut être par rapport à un autre récipient plus volumineux, égal ou moins volumineux. Il faut donc un deuxième élément qui est le suivant.

2) L'enfant commence vers le deuxième stade à partitionner ou à décomposer en unités égales. Il faut qu'une partition quelconque double la mise en relation.

---

[2] L'addition concerne des relations de même type et consiste en la coordination de rapports selon une seule dimension. Exemple : A>B et B>C alors A>C.
La multiplication concerne des relations de types différents et consiste en la coordination de rapports du point de vue de deux ou plusieurs relations(dimension).

Une dernière question subsiste à laquelle nous allons répondre brièvement.

Comment se fait-il qu'il faille attendre le troisième stade pour que la correspondance terme à terme (CTT) entraîne l'équivalence durable des collections, tandis que durant les deux premières périodes elle ne suffit point à vaincre les apparences perceptives?

Remarque : Un exemple de correspondance terme à terme est donné dans la section "Une première approche" (comparaison de 2 collections d'anneaux).

La raison est que la correspondance elle-même ne se construit pas de manière spontanée mais progressivement. Elle évolue également en 3 stades qui sont les suivants :

- Stade 1. : Ni correspondance exacte, ni équivalence.

  L'enfant de ce stade procède par simple correspondance qualitative globale fondée sur sa seule perception :

  Comparaison globale des longueurs des collections.

  La CTT n'est pas quantifiante. Les qualités perçues par l'enfant ne donnent lieu qu'à de simples rapports qualitatifs (plus ou moins 'grand', 'long', ...) sans opération proprement dite. En effet, ces qualités ne sont pas coordonnées entre elles. C'est cet aspect non opératoire c'est-à-dire la non réversibilité des évaluations de ce stade qui explique leur échec. Il n'y a même pas de correspondance intuitive (optique).

- Stade 2. : CTT mais sans équivalence durable entre les collections correspondantes.

  Il croit à l'équivalence lorsqu'il se trouve en présence d'une correspondance visuelle entre les deux collections mais il cesse de croire à cette équivalence dès que l'on sépare les couples de termes corrélatifs en espaçant ou en resserrant les termes de l'une des deux collections.

  Tout se passe comme si la quantité dépendait moins du nombre ou de la CTT entre objets discrets que de l'aspect global de la collection (ou encore des particularités qualitatives) et en particulier de l'espace occupé. Il y a à ce niveau coordination mais uniquement sur un plan intuitif.

- Stade 3. : CTT et équivalence durable des collections correspondantes.

  Les ensembles une fois mis en correspondance biunivoque et réciproque et ainsi rendus équivalents au moment de cette correspondance, le demeurent ensuite quel que soit l'arrangement de leurs éléments. L'opération de la mise en correspondance biunivoque et réciproque est ainsi constituée par delà la comparaison simplement intuitive (optique). La correspondance est affranchie

de l'intuition et l'enfant atteint par cela même la réversibilité et l'équivalence. L'enfant découvre que toute transformation spaciale dans la disposition des éléments peut être corrigée par une opération inverse. Ce stade marque le primat de l'opération proprement dite sur la perception. La correspondance est dite 'quantifiante'.

D) La notion de quantité.

Nous évoquerons d'abord la notion de quantité selon Piaget. Ensuite, nous irons au-delà de Piaget et nous donnerons une définition plus récente et nous terminerons par une comparaison de ces définitions.

La quantité selon Piaget.

Pour Piaget, la quantité est une notion tardive résultant de l'intériorisation d'actions et de la coordination de ces actions. Piaget utilise comme critère d'acquisition de la quantité le fait que les enfants soient 'conservants'.
Piaget interprète le moment où l'enfant conserve l'extension de la collection (quand il maintient son jugement initial d'équivalence), comme ce moment du développement où 'la correspondance terme à terme devient réellement quantifiante'. L'acquisition de la notion de quantité de Piaget dépend seulement du degré de développement de l'enfant.

D'autre part, Piaget ne distingue pas les notions de quantité et de nombre.

Au-delà de Piaget (Remi Brissiaud et ses partisans).

Remi Brissiaud et ses partisans définissent la quantité comme représentation symbolique de l'extension des collections.

Explicitons quelque peu cette définition.

La quantité est la mesure d'une propriété des collections. Quelle propriété des collections la quantité mesure-t-elle ?

Prenons comme base de l'explication une analogie entre une tige de fer et une quantité.

Il est possible de mesurer la longueur d'une tige de fer en se fixant au préalable une unité. Mais quelle sorte de propriété est la longueur ? En fait, on peut parler de la longueur d'une tige par ce qu'il existe un moyen physique de comparaison des tiges selon cette caractéristique : la juxtaposition. En les juxtaposant, il est possible de dire

si plusieurs d'entre elles ont la même longueur et de dégager ainsi la notion de longueur : Des tiges qui coïncident aux extrémités après juxtaposition et quelles que soient leurs autres caractéristiques sont considérées comme équivalentes du point de vue de leur longueur. Pour les collections, le procédé de comparaison, correspondant à la juxtaposition pour les tiges, est la correspondance terme à terme.

En résumé, on a la situation suivante :

	Tige		Collection	
Juxtaposition	Longueur	⟺	?	Correspondance t. à t.
	Mesure		Quantité	

Quelle propriété des collections la quantité mesure-t-elle ?

Cette question est équivalente à la suivante : Quand 2 tiges coïncident en les juxtaposant, on dit qu'elles ont même longueur, quand 2 collections peuvent être mises en correspondance terme à terme, quelle sorte de grandeur ont-elles en commun ?

On dit que 2 collections en correspondance terme à terme ont même extension c'est-à-dire que la quantité est la mesure de l'extension d'une collection.

On a donc les correspondances suivantes :

	Tige		Collection	
Juxtaposition	Longueur	⟺	?	Correspondance t. à t.
	Mesure		Quantité	

La quantité est donc la mesure de l'extension des collections.

Mais pour décider que 2 collections ont la même extension, il n'est pas toujours nécessaire de les quantifier. Une juxtaposition dans le cas de tiges ou une correspondance terme à terme dans le cas de collections est suffisante pour les comparer. Il est superflu d'en prendre la mesure.

Cependant, il faut différencier cette situation de comparaison de 2 collections simultanément présentes de celle où il s'agit de communiquer quelle est l'extension d'une collection.

Si la quantification n'est pas nécessaire pour répondre à des questions du genre "où est-ce qu'il y a le plus ?", elle l'est pour répondre aux questions du type "Combien il y a ...".

Rappelons-nous du berger qui construisait une collection de cailloux ayant la même extension que son troupeau. Ce berger ne procédait pas à une simple comparaison, car il n'accordait pas le même statut aux 2 collections : l'une servait à représenter l'extension de l'autre. Il procédait de la sorte afin de savoir "combien il y a de bêtes".

On passe d'une simple comparaison de l'extension des collections à leur quantification en brisant l'égalité de statut de toutes les collections, en accordant un statut spécifique à certaines d'entre elles : celles-ci deviennent des collections-témoins. Elles représentent non seulement leur extension mais également l'extension de toute collection équivalente.

On peut donc dire que les collections de cailloux permettaient aux bergers de concevoir les quantités.

De même on dira qu'un enfant possède une première conception des quantités dès qu'il dispose d'un système symbolique qui lui permet de communiquer concernant l'extension des collections.

Comparaison entre l'approche Piagetienne de la quantité et celle de Brissiaud.

Critère d'acquisition de la quantité par l'enfant.	
Piaget	Brissiaud
Le fait que les enfants soient 'conservants'.	Le fait que l'enfant dispose d'un système symbolique qui lui permet de communiquer concernant l'extension de collections.

Piaget, contrairement à Brissiaud, considère donc le fait qu'une correspondance terme à terme soit quantifiante comme une propriété qui dépend seulement du degré de développement de l'enfant. Pour Brissiaud, la nature quantifiante d'une correspondance t. à t. dépend aussi de la situation de communication dans laquelle cette correspondance est mise en oeuvre.

Le rôle attribué aux représentations symboliques dans le processus de progrès.	
Brissiaud	Piaget
Il pense que l'enfant progresse vers une bonne conception de l'extension des collections selon un processus qui a deux composantes : - Une composante pratique, dans la mesure où le progrès trouve sa source dans les actions du sujet, lors des résolutions pratiques qui utilisent la correspondance terme à terme. - Une composante symbolique : l'usage de représentations symboliques c'est-à-dire la quantification . Cet usage repose sur des actions (construire sur une collection-témoin, compter), mais ces actions ne peuvent être assimilées à celles qui utilisent un matériau quelconque, dans la mesure où l'aspect symbolique des collections témoins et des mots-nombres donne à leur usage des significations qui interfèrent avec les significations issues de l'action.	Piaget ne distingue pas ces deux sortes d'actions car il pense que les instruments symboliques tel que le langage ne peuvent pas être la source du progrès.

Les conséquences.	
Brissiaud	Piaget
Il accorde de manière plus précoce une première conception des quantités à l'enfant.	La quantité est une notion tardive.
Il permet une première approche du nombre assez tôt.	Il préconise de ne pas enseigner la quantité ou le nombre tant que les enfants n'ont pas atteint la conservation de l'extension des collections.

Conclusion.

Selon Brissiaud, un enfant qui sait utiliser un système symbolique (le plus connu étant les doigts des mains) pour communiquer avec son entourage possède déjà une première conception des quantités.
Cette définition est différente de celle de Piaget, pour qui la quantité est une notion tardive, qui résulte de l'intériorisation d'actions et de la coordination de ces actions intériorisées.

Fondamentalement, cette différence de définition résulte de conceptions du progrès qui n'accordent pas le même statut aux représentations symboliques et donc à la communication avec les adultes.

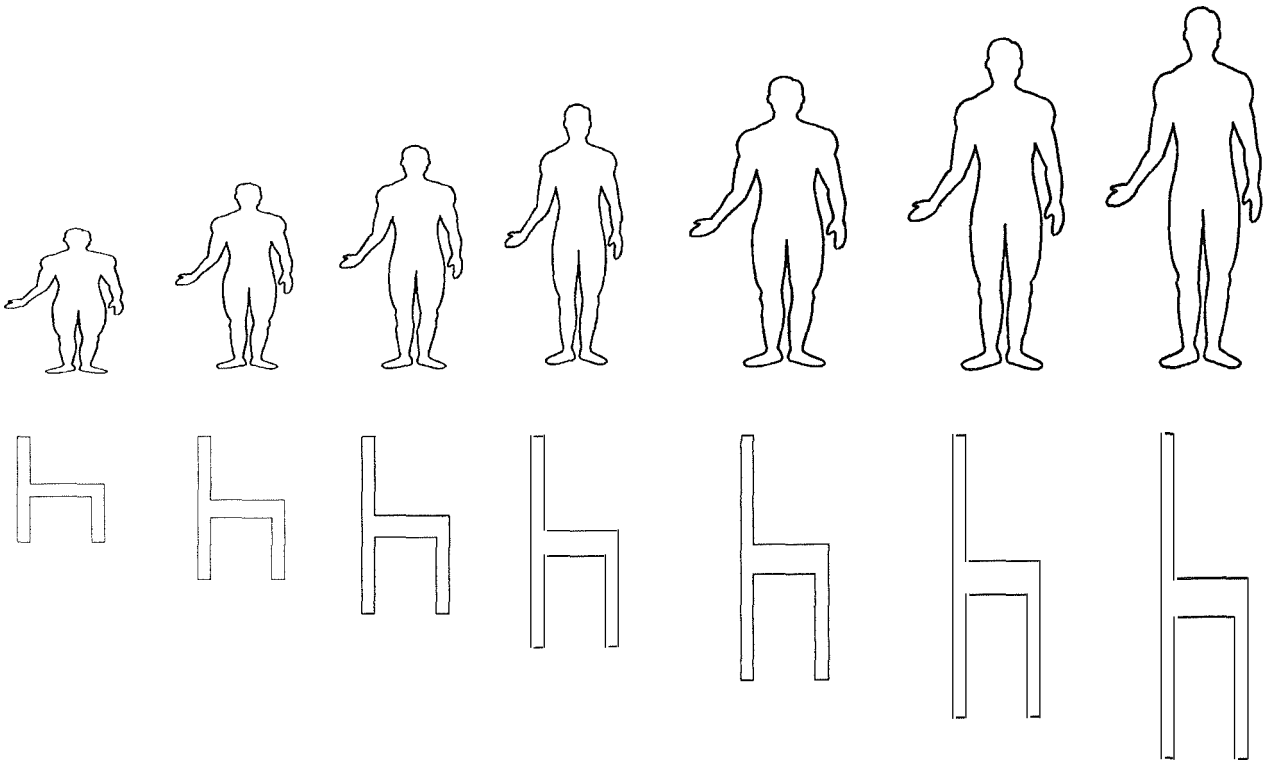
E) Autres apprentissages étroitement liés à la conservation et évolution de l'enfant dans ces apprentissages.

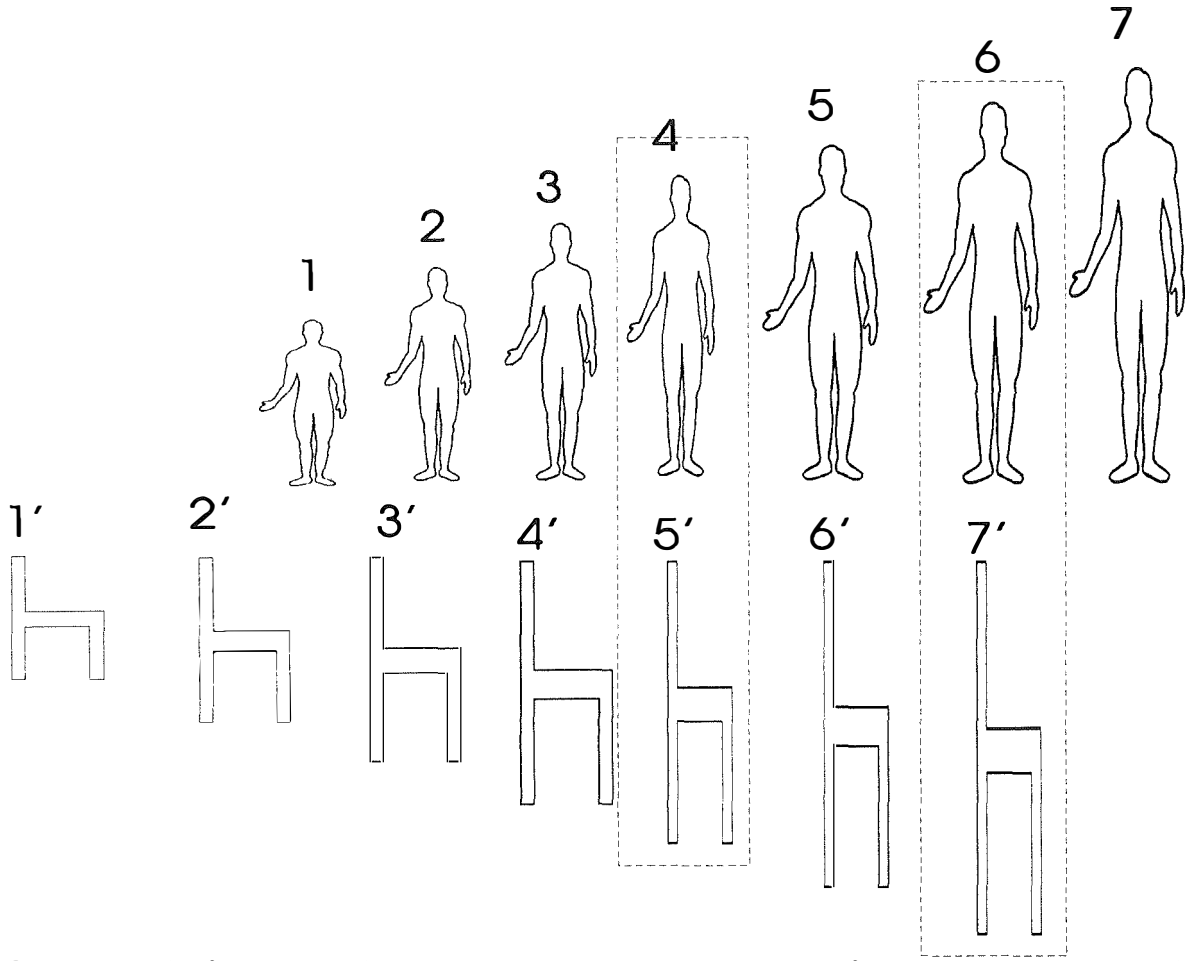
L'inclusion ordinale (ou encore : sériation).

Qu'est ce que la sériation ?

La sériation traite de l'ordonnancement. Il s'agit de faire comprendre à l'enfant la notion d'ordre.

Exemple d'épreuve de sériation.





On demande à l'enfant de trouver la chaise qui convient à un bonhomme que l'on désigne.
Pour un enfant de 3-4 ans :

- L'enfant indique automatiquement et quelle que soit sa taille, la chaise se trouvant devant le bonhomme désigné.

- C'est donc une similitude d'emplacement qui constitue le seul point de repère de l'enfant.

On a donc une correspondance optique uniquement : comparaison globale sans compréhension du détail des rapports.

Pour un enfant de 5-6 ans :

L'enfant comprend la signification de la correspondance entre les deux séries.

Deux procédés pour arriver à une solution correcte :

a) Procédé rudimentaire

L'enfant utilise par exemple les deux mains pour progresser simultanément entre les deux séries.

b) Procédé plus évolué :

L'enfant comprend que l'élément désigné et l'élément à trouver ont une position similaire dans les deux séries et qu'il peut déterminer cette position en comptant dans l'une et l'autre série.
Le problème est que la plupart du temps l'enfant se trompe, il choisit l'élément qui précède l'élément correct. En fait, une fois un élément désigné, le problème que perçoit l'enfant est celui de déterminer le nombre d'éléments qui le précèdent. Mais lorsqu'il transpose ce résultat dans l'autre série, il n'y a aucun point de repère qui lui rappelle qu'il avait décomposé en << ce qui précède >> et << l'élément désigné >>.

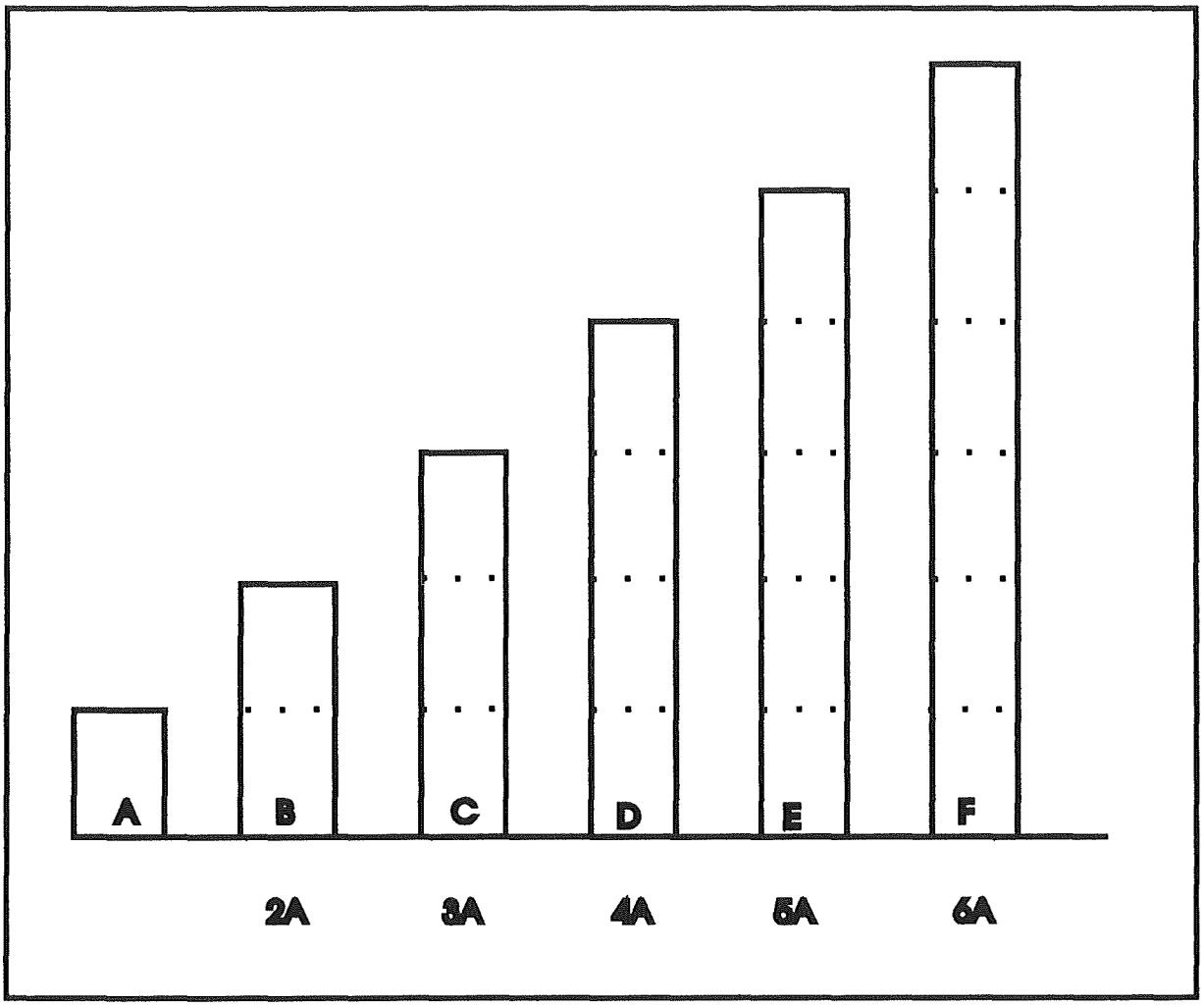
La relation entre ordination et cardination est encore insuffisante. L'enfant n'évalue pas la place d'un élément par son numéro d'ordre. Il s'établit dans l'esprit de l'enfant une dissociation entre le rang désigné et la collection des termes précédents qui pour lui n'ont pas la même fonction

Vers 7 ans :

Les relations entre ordination et cardination sont correctement établies.
Dans deux séries correspondantes de dix éléments chacune, le 7^{e} (ordinal) dans l'ordre croissant est le dernier des sept (cardinal) éléments et est en même temps, le 4^{e} (ordinal) élément pris dans l'ordre décroissant. Il coordonne la détermination du rang recherché avec celle de la valeur cardinale des collections intéressées.

Les relations entre ordination et cardination.

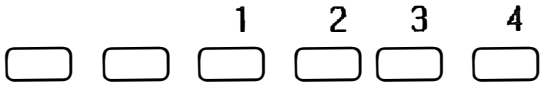
Une technique généralement utilisée pour faire comprendre aux enfants les relations entre ordinal et cardinal est l'épreuve des rectangles :



On demande combien d'unités on peut faire avec un rectangle désigné.

On cherche à faire découvrir à l'enfant que la valeur en unités d'une marche est égale à la somme des marches à parcourir jusqu'à et y compris celle qui est désignée.

Cette situation est une situation privilégiée pour aider l'enfant à comprendre la relation entre l'ordinal et le cardinal. Néanmoins, le comptage de gauche à droit peut débuter au sein d'une série d'objets :



Il faut remarquer aussi que le comptage peut se réaliser de la droite vers la gauche.

A 5 ans.

La relation entre l'ordre et la cardination n'est pas comprise.

Il essaie d'évaluer combien d'unités peut contenir la marche désignée.

A 6 ans.

L'enfant a une compréhension ordinale de la série. Elle est liée à l'acte intuitif au moyen duquel il est possible de parcourir la série terme à terme du commencement à la fin.

A 7ans, l'enfant comprend les relations entre :

- le numéro d'ordre d'un élément.
- le nombre d'éléments précédents que ce numéro d'ordre signifie et
- le nombre d'unités constituantes que ce numéro d'ordre implique.

Il est donc capable de faire correspondre d'emblée la valeur cardinale d'un rectangle à son rang.

Le calcul unitaire.

De quoi s'agit-il ?

On parle de calcul unitaire, lorsque l'enfant compte unité par unité. Il utilise, par exemple, ses doigts un à un.

Il faut tenter de faire disparaître le plus rapidement possible ce mode de calcul.

Les groupements ou encore l'inclusion cardinale (composition/décomposition)

Trois grandes étapes dans les groupements.

A) Le dénombrement

De quoi s'agit-il ?

On dit qu'un enfant sait dénombrer une collection quand le dernier mot-nombre qu'il prononce (ou écrit) n'est pas un simple numéro, mais représente à lui seul la quantité de tous les objets.

Pourquoi la transition du comptage-numérotage au dénombrement est-elle si difficile?

On parle de comptage-numérotage lorsque le comptage fonctionne comme une attribution de dossards à des coureurs. Il sait que lorsqu'on va plus loin dans la distribution des dossards, ça signifie qu'il y a plus d'objets. Mais aucun mot-nombre prononcé ne représente à lui tout seul une quantité. Le dernier mot-nombre ne vaut pas plus que les autres. Ce n'est qu'un numéro qui réfère l'objet pointé. C'est cette forme de comptage qu'on appelle un comptage-numérotage. Chacun des mots-nombres prononcés (écrits), y compris le dernier, est un numéro qui réfère uniquement à l'objet pointé. Le mot-nombre n'a qu'un aspect ordinal.

Le passage du comptage-numérotage au dénombrement, une transition difficile. Pourquoi ?

Pour accéder au dénombrement, à partir du comptage-numérotage, l'enfant doit accorder une double signification au dernier mot-nombre prononcé : lorsqu'il est prononcé pour la première fois, au cours du comptage, le dernier mot-nombre a le même sens que tous les autres mots-nombres, c'est un numéro qui distingue un objet. L'enfant doit alors changer la signification de ce mot-nombre pour qu'il représente une quantité de tous les objets : on passe de "le sept" à "les sept".

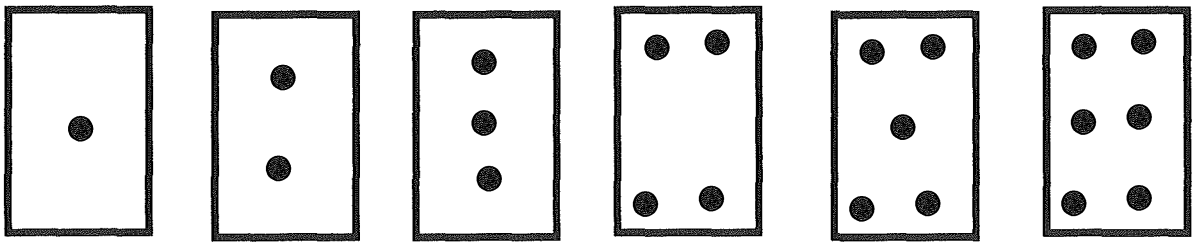
L'usage de constellations facilite l'accès au dénombrement.

Qu'est - ce que des constellations ?

Il s'agit de configurations de points représentant des quantités et qui facilitent la reconnaissance de ces quantités. Rapidement, les enfants savent les nommer. Mais attention, quand les enfants nomment une constellation, ce n'est pas la quantité correspondante qu'il désigne, car une constellation correspond à une configuration spatiale bien déterminée (alors que la quantité est invariante quand on change la configuration spatiale).

En fait, les schèmes (constellations) sont un matériel symbolique permettant une reconnaissance immédiate et qu'il faut fournir aux enfants .

Les schèmes classiques sont les suivants :



En jouant par exemple avec des dés, l'enfant a la possibilité de prendre conscience qu'un même mot-nombre peut signifier à la fois un numéro et une constellation (une quantité).

Pour aider un enfant à accéder au dénombrement, les propositions suivantes ont été avancées :

- Mettre en place des activités où c'est l'adulte qui récite (ou écrit) la comptine numérique alors que la tâche de l'enfant est de pointer les objets correspondants.

- Mettre en place des jeux de dés et intervenir dans le jeu de l'enfant pour l'aider à prendre conscience de la double signification des mots-nombres qui désignent les constellations.

B) Reconnaître globalement des images (schèmes).

Il s'agit ici d'entraîner les enfants à reconnaître un nombre à partir d'une représentation.

C) Opérer sur les groupements : calcul et composition/décomposition

Qu'est-ce que le calcul ?

Calculer, c'est mettre en relation des quantités, directement à partir de leurs représentations numériques, sans passer par la réalisation physique d'une ou plusieurs collections dont les éléments seraient dénombrés.

Calcul versus comptage.

Pour déterminer une somme par une procédure de comptage, les enfants représentent chaque quantité par une collection-témoin d'objets avant de recompter l'ensemble des objets.
En revanche, l'enfant qui calcule, dira directement "neuf et quatre, ça fait treize".

L'apprentissage du calcul à l'aide de collections-témoins organisées (CTO).

Qu'est-ce que les collections-témoins organisées ?

Ce sont des collections-témoins qui permettent une représentation rapide de la quantité correspondante, grâce à la configuration spatiale qui leur est associée.

Exemple de CTO : Les constellations, les configurations de doigts, les réglettes cuisenaires, ...

Il ne fait guère de doute que le développement de bonnes compétences numériques nécessite l'usage de CTO.

Qu'est ce que la décomposition ?

Il s'agit de décomposer un tas d'éléments (et donc un nombre) en un certain nombre de tas de taille plus petite ou encore de demander aux enfants de remplir des égalités lacunaires élémentaires où seul l'addition est utilisée, l'inconnue étant l'opérande d'un opérateur d'addition.

Exemple :

Faire des tas de 3 pions à partir de 12 pions.

Décomposer en deux tas dont 1 a au moins 3 pions.

ou encore

5 + ... = 7
7 + ... = 8

La Composition.

La composition est l'opération inverse de la décomposition. Il s'agit de faire un seul tas à partir de groupements.
Ici, on demande généralement aux enfants de remplir des égalités lacunaires dont l'inconnue est le résultat.

Exemple :

5 + 2 = ...
3 + 3 = ...

Remarque :

Il est évident que pour pouvoir réaliser composition/décomposition, l'enfant devra avoir acquis la signification de l'addition ainsi que du symbole spécifique '+'.

Partie 3 : ANALYSE INFORMATIQUE.

8. Introduction.

Lors de la lecture de l'analyse informatique, il convient de toujours bien garder à l'esprit que les différents développements ont eu lieu dans le cadre de la notion de conservation du nombre (plus précisément d'ensembles numériques).

Quels sont les concepts intervenant lors de l'apprentissage de la notion de conservation du nombre ?

Nous avons les concepts suivants :

- L'objet : La bouteille, le vase, la fleur, la bille, l'oeuf, la pomme, le raisin, ...
- Le tas ou l'ensemble qui se compose généralement d'objets semblables c'est-à-dire de même nature, de même taille, de même aspect.
- La constellation : Représentation imagée d'une quantité - d'un nombre.
- L'enfant.
- L'instituteur.

Puisque nous rencontrons ces concepts lors de l'apprentissage réel (par opposition à virtuel c'est-à-dire à l'aide de l'ordinateur, sans matérialisation de certains de ces concepts) de la conservation, il nous semble intéressant de développer un type objet pour chacun de ces concepts.

Nous aurons donc les correspondances suivantes :

Concepts.	Types objets.
Objet	TElement
Constellation	TConstellation
Tas ou ensemble	TTas
Enfant	TUtilisateur
Instituteur	TGerant

Afin d'éviter toute ambiguïté sur le terme objet, nous avons préféré appeler un objet d'un tas 'un élément'.

9. Fiche type utilisée pour la spécification d'objets.

Ce chapitre décrit le type de fiche utilisée pour spécifier des types objets.
Certaines parties seront omises selon le type spécifié.

Type objet :

Indique le nom du type que l'on spécifie.

Champs.

Cette section reprend les champs propres au type objet. Par champs propres, on entend les champs qui lui appartiennent mais qui n'appartiennent pas à son ancêtre.

Nom	Type	Signification

Méthodes.

Description des méthodes propres à l'objet.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Remarques :

Destructeur : **Done**

But de la méthode : détruire les variables dynamiques qui dépendent de l'objet.

Remarques :

Nom de la méthode :

Indique le nom de la méthode.

Champs modifiés :

Indique les champs qui sont modifiés.

Les champs propres au type objet apparaîtront normalement.
Les champs hérités apparaîtrons entre parenthèses.

La syntaxe sera la suivante : Champ → Nouvelle valeur ou Opération.

Si 'Nouvelle valeur' ou 'Opération' est entre crochets [], cela signifie que la modification est conditionnelle. Dans le cas où la nouvelle valeur n'est pas déterminée précisément, l'expression 'Nouvelle valeur' sera indiquée.

Exemple : (Nombre → [+1]) signifie que le champ hérité Nombre est augmenté de 1 sous certaines conditions. Vous trouverez ces conditions dans la section 'Valeur retournée par la méthode' ou 'But de la méthode'.

Si le type spécifié est ou comporte une collection, nous parlerons de modification de la collection, d'un élément de la collection, ... Nous vous renvoyons à la section relative aux collections pour plus de détails.

Paramètres formels non modifiés :

Indique les paramètres de la méthode qui ne sont pas modifiés par la méthode.

Nom du paramètre	*Type*	*Signification*

Paramètres formels produits ou modifiés :

Indique les paramètres de la méthode qui sont modifiés par la méthode.

Nom du paramètre	*Type*	*Signification*

Valeur retournée par la fonction :

Type :

Signification :

But de la méthode :

Dans cette partie, le nom d'un paramètre de la méthode sera souligné tandis que le nom d'un champ propre à l'objet spécifié apparaîtra en italique.

10. Conventions utilisées.

Le nom d'un type non-prédéfini autre qu'un pointeur commence par la lettre T.

Un type énuméré commence par TE.

Le nom d'un type pointeur commence par la lettre p et est suivi par le nom du type pointé duquel on a retiré le préfixe T (pour autant que celui-ci soit présent).

La partie suivant le T ou le P comprend un ou plusieurs mots français ou abréviations de mots français.

Seules les constantes pré définies par Windows ou BP (Borland Pascal) et les constantes identifiant une icône-élément apparaîtrons en majuscule.

Tout type, variable, procédure, fonction ou méthode pré défini (et non spécialisé) ne comprendra que des minuscules.

Tout type, constante, variable, procédure, fonction ou méthode pré défini et spécialisé ou non pré défini commencera par une majuscule.

Toute constante non pré définie commencera par un C .

Exceptions :

- Les constantes de recensement commenceront par R suivi du nom du type recensé duquel on a retiré le préfixe T

- Les constantes relatives aux ressources.

Toute procédure non pré définie commencera par un P.

Toute fonction non pré définie commencera par un F.

Toute méthode non pré définie commencera par un M suivi d'un P ou d'un F selon qu'il s'agit d'une procédure ou d'une fonction.

Des abréviations seront utilisées pour les mots couramment utilisés (ex : Coord pour Coordonnées).

Une constante d'un type énuméré commencera par la lettre E.

Les méthodes gardant un nom anglais sont généralement des méthodes virtuelles d'objets fournis par **Borland Pascal 7.0.**

Les paramètres pré définis (non - pré définis) de méthodes commencent par une minuscule (majuscule).

Les champs pré définis (non pré définis) des objets commencent par une minuscule (majuscule).

11. Les collections

Les objets développés font souvent référence au concept de collection. Pour cette raison, nous en faisons une brève description.

Qu'est-ce qu'une collection ?

Une collection est une instance du type *tcollection* fourni par **ObjectWindows**. Nous vous présentons ses champs ultérieurement dans ce chapitre et vous trouverez ses méthodes aux pages 375-381 du '*Guide du programmeur*' **d'ObjectWindows**. Son but principal est de permettre de stocker plusieurs données.

Quelles sont les caractéristiques d'une collection ?

- Un dimensionnement dynamique.

  Contrairement au tableau standard, une collection n'a pas une taille fixe déterminée lors de la compilation. Vous lui spécifiez une taille initiale lors de la compilation mais elle peut croître dynamiquement lors de l'exécution afin de recevoir les données qu'elle stocke.

- Le polymorphisme.

  Une collection est polymorphe c'est-à-dire qu'elle peut contenir des données de types différents ce qui n'est pas le cas des tableaux standards.

En fait, une collection ne stocke pas la donnée elle-même mais un pointeur sur cette donnée. Ce n'est donc pas vraiment une collection de données mais plutôt une collection de pointeurs sur des données. Ces pointeurs sont des pointeurs sans type, c'est ce qui permet de stocker des données de différents types. Sauf spécification contraire, une collection ne peut stocker que des pointeurs vers des objets d'un type descendant du type objet *tobject* fourni par **ObjectWindows**.

En résumé, nous dirons qu'une collection est un tableau extensible de pointeurs, chaque pointeur pointant normalement sur une instance d'un type objet descendant du type objet *tobject* défini par **Borland Pascal**.

Les champs du type *tcollection*.

count : integer

Indique le nombre d'éléments de la collection, au maximum MAXCOLLECTIONSIZE (constante définie par Borland Pascal).

delta : integer;

Valeur à ajouter à la taille de la liste items à chaque fois qu'elle est pleine. Si delta vaut 0, la collection ne peut être étendue au-delà de la taille indiquée dans limit.

items : pitemlist

Il s'agit d'un pointeur sur un tableau de pointeurs qui pointent normalement sur des instances de types objets descendant du type objet *tobject*. C'est la composition de la collection.

limit : integer

Taille courante allouée(en nombre d'éléments) de la liste items.

Langage adopté.

Afin d'éviter toute ambiguïté entre le terme 'objet' dans le sens de type objet, 'objet' dans le sens instance d'un type objet, le terme 'objet de la collection' dans le sens 'membre de la collection' et 'objet' dans le sens d'une chose matérielle, nous utiliserons le terme 'élément de la collection' pour désigner un membre de la collection que ce soit un objet ou non. Toutefois, il se peut que pour certaines collections, le terme 'élément' soit remplacé par un terme plus approprié (ex : si une collection ne contient que des pointeurs vers des instances du type *octet*, le terme 'élément' peut être remplacé par le terme 'octet'.

Remarque : Vu ce qui a été dit dans l'introduction, dans le cas où l'élément d'une collection sera du type TElement, le terme élément aura donc une double signification:

- Celle de membre de la collection.
- Celle d'objet d'un tas.

D'autre part, afin de simplifier les phrases, et sachant que PPPPPP est un pointeur et CCCCCC un pointeur sur une collection, nous parlerons :

de	plutôt que de
l'élément PPPPPP	l'élément pointé par le pointeur (variable, paramètre, champ) PPPPPP.
la collection CCCCCC	la collection pointée par le pointeur (v,p,c) CCCCCC.
l'élément PPPPPP de la collection CCCCCC.	l'élément pointé par le pointeur PPPPPP contenu dans la collection pointée par le pointeur CCCCCC.
un élément de la collection	un pointeur de la collection pointant sur un élément.
un élément vide	un pointeur valant Nil.

D'autre part, en utilisant déjà les conventions citées précédemment, nous dirons

que	si
certains (tous) éléments de la collection CCCCCC sont modifiés	certains (tous) éléments de la collection CCCCCC ont au moins un de leurs champs modifiés.
la collection CCCCCC est modifiée	l'un des champs de la collection CCCCCC est modifié.
le pointeur PPPPPP ou CCCCCC (la variable, le paramètre ou le champ) est modifié	la valeur du pointeur PPPPPP ou CCCCCC (la variable, paramètre ou champ) est modifiée.
l'élément PPPPPP appartient à la collection CCCCCC	si la valeur du pointeur PPPPPP fait partie des pointeurs de la collection CCCCCC.[3]

Sachant que TTYPE est un type quelconque et PTYPE un type pointeur sur une instance du type TYPE, nous parlerons :

de	plutôt que de
l'élément du type TTYPE de la collection	l'élément du type TTYPE pointé par un pointeur du type PTYPE de la collection.

[3] C'est la valeur du pointeur et non la valeur de l'instance qui doit appartenir à la collection. Exemple : Supposons qu'on ait une collection d'entiers. Parmi les entiers de la collection, on a la valeur 5. Le fait d'avoir une variable dynamique de type entier de valeur 5 pointée par le pointeur PE ne signifie pas pour autant que PE appartient à la collection. PE n'appartient à la collection que si la variable dynamique pointée par PE est la même que celle de la collection c'est-à-dire si PE est égal au pointeur de la collection pointant la variable dynamique dont la valeur est 5.

12. Module 'Globaux'.

A) Spécifications.

1) Types objets.

Type objet : THistoriqueOctets

Champs.

Nom	Type	Signification
NbrOctetsMaximum	byte	Nombre maximum d'éléments (pour cette collection, nous parlerons d'octets) que la collection peut contenir.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Nom de la méthode : **MPAjouter**

Champs modifiés : La collection : (count → [+1]) et éventuellement (limit → +delta)

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnOctet	byte	La valeur à ajouter dans la collection.

But de la méthode :

Ajouter à la collection un pointeur sur un octet dont la valeur est le paramètre UnOctet. Si le nombre d'octets de la collection est égal au nombre maximum d'octets que peut contenir la collection, l'octet le plus ancien (c'est-à-dire le premier) est supprimé et détruit avant l'insertion de la nouvelle valeur.

Nom de la méthode : **FreeItem**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
item	pointer	Pointeur sur l'octet de la collection à libérer.

But de la méthode :

Détruire l'octet pointé par item.

Nom de la méthode : **MFContient**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnOctet	byte	La valeur à rechercher parmi les octets de la collection.

Valeur retournée par la méthode :

Type : boolean

Signification : Prend la valeur TRUE si un des octets de la collection est égal à la valeur UnOctet et FALSE dans le cas contraire.

Type objet : TValidateurIntervalleReels

Champs.

Nom	Type	Signification
Max	real	Réel le plus grand autorisé.
Min	real	Réel le plus petit autorisé.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Destructeur : **Done**

But de la méthode : détruire les variables dynamiques qui dépendent de l'objet.

Constructor : **Load**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un Flux.

But de la méthode :

Construire et charger une instance du type spécifié à partir du flux Flux.

Nom de la méthode : **Store**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Flux dans lequel l'instance sera enregistrée.

But de la méthode :

Enregistrer une instance du type spécifié dans le flux Flux.

Nom de la méthode : **Error**

But de la méthode :

Afficher un message d'erreur indiquant que la valeur entrée n'appartient pas à l'intervalle spécifié.

Nom de la méthode : **IsValid**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
s	string	Un nombre réel sous forme de chaîne de caractères.

Valeur retournée par la méthode :

Type : boolean

Signification : Vaut TRUE si le résultat vérifie les trois conditions :

- C'est un nombre réel valide
- Sa valeur est supérieure ou égale à Min
- Sa valeur est inférieure ou égale à Max.

But de la méthode :

Convertir une chaîne de caractères s en un nombre entier.

Type objet : TValidateurIntervalleEntiers

Nom de la méthode : **Error**

But de la méthode :

Afficher un message d'erreur indiquant que la valeur entrée n'appartient pas à l'intervalle spécifié.

Type objet : TUneValeurResultat

Champs.

Nom	Type	Signification
Valeur	boolean	Valeur booléenne indiquant un échec ou une réussite.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Constructeur : **Load**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un flux.

But de la méthode :

Construire et charger une instance du type TUneValeurResultat à partir du flux Flux.

Nom de la méthode : **Store**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un Flux.

But de la méthode :

Enregistrer une instance du type TUneValeurResultat dans le flux Flux.

Type objet : TBoolCollection

Champs.

Nom	Type	Signification
NbrMaximumResultats	word	Nombre maximum d'instances du type TUneValeurResultats que la collection peut contenir.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Constructeur : **Load**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un flux.

But de la méthode :

Construire et charger une instance du type TBoolCollection à partir du flux Flux.

Nom de la méthode : **Store**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un Flux.

But de la méthode :

Enregistrer une instance du type TBoolCollection dans le flux Flux.

Nom de la méthode : **MFAjouter**

Champs Modifiés : La collection : (count → [+1]) et éventuellement (limit → +delta)

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UneValeurResultat	PUneValeurResultat	Instance à ajouter à la fin de la collection.
Suppression	boolean	Indique si le premier élément de la collection doit être supprimé et libéré dans le cas où le nombre maximum d'éléments que la collection peut contenir est déjà atteint.

Valeur retournée par la méthode :

Type : boolean

Signification : Vaut TRUE si l'ajout a eu lieu.

But de la méthode :

Ajouter en fin de collection, l'élément UneValeurResultat.

Si au moment d'ajouter UneValeurRésultat, le nombre d'éléments de la collection est égal à *NbrMaximumResultats*, le paramètre Suppression détermine si l'ajout aura effectivement lieu. Si Suppression vaut TRUE, le premier élément de la collection sera supprimé et libéré afin de pouvoir ajouter UneValeurResultat et la fonction renverra TRUE. Dans le cas contraire, la collection est maintenue dans son état initial et la fonction renvoie FALSE.

Nom de la méthode : **MFRemplir**

Champs Modifiés : La collection : (count → [NbrMaximumResultats]) et éventuellement (limit → +delta).

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
NbrFALSE	byte	Nombre de pointeurs, pointant sur des instances du type TUneValeurResultat avec le champ valeur valant FALSE, à ajouter à la collection.

Valeur retournée par la méthode :

Type : Boolean

Signification : Vaut TRUE si le remplissage a eu lieu, FALSE dans le cas contraire.

But de la méthode :

Remplir la collection avec *NbrMaximumResultats* éléments du type TUneValeurResultat. Sur ces *NbrMaximumResultats* éléments, NbrFALSE éléments auront leur champ Valeur à FALSE. Les autres éléments auront leur champ Valeur à TRUE. Si NbrFALSE est supérieur à *NbrMaximumResultats*, la méthode est sans effet.

Remarque :

Si la collection comprend déjà des éléments, ceux-ci sont supprimés et libérés avant le remplissage.

Nom de la méthode : **MFParcourir**

Valeur retournée par la méthode :

Type : integer

Signification : Nombre d'éléments de la collection dont le champ Valeur vaut FALSE.

Type objet : TResultats

Champs.

Nom	Type	Signification
TableauResultats	array[1..10] of PBoolCollection	Tableau permettant de stocker un certain nombres de collections d'éléments du type TUneValeurResultats.
LongTableauResultats	byte	Longueur effective du tableau TableauResultats

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Remarques :

Les LongTabResultats premières collections du tableau TableauResultats sont créées et initialisées avec la valeur 5 comme valeur de leur champ limit, delta et NbrMaximumResultats. Les autres collections du tableau TableauResultats valent nil.

Constructeur : **Load**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un flux.

But de la méthode :

Construire et charger une instance du type TResultatsComparaisons à partir du flux Flux.

Nom de la méthode : **Store**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un Flux.

But de la méthode :

Enregistrer une instance du type TResultats dans le flux Flux.

Nom de la méthode : **Done**

But de la méthode : Détruire les variables dynamiques dépendant de l'objet.

Type objet : TCollectionPoints

Nom de la méthode : **FreeItem**;

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
item	pointer	Pointeur sur l'élément à détruire.

But de la méthode :

Détruire l'élément item.

2) Fonctions diverses.

Nom de la procédure : **PCreerBrosseSolide**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Couleur	longint	Couleur de la brosse.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
UneBrosse	hbrush	Outil brosse.

But de la procédure :

Créer un outil brosse Windows UneBrosse dont la couleur est donnée par le paramètre Couleur. Si lors de l'appel, le paramètre UneBrosse désigne déjà un outil brosse (UneBrosse est <> de 0), la brosse courante est supprimée avant d'être remplacée par la nouvelle.

Nom de la procédure : **PCreerPlumeSolide**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Couleur	longint	Couleur de la plume.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
UnePlume	hpen	Outil plume.

But de la procédure :

Créer un outil plume (crayon) Windows UnePlume dont la couleur est donnée par le paramètre Couleur. Si lors de l'appel, le paramètre UnePlume désigne déjà un outil plume (UnePlume est <> de 0), la plume courante est supprimée avant d'être remplacée par la nouvelle.

Nom de la procédure : **PRemplirDe**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Taille	word	Taille de la variable VariableARemplir.
ValeurDeRemplissage	char	Caractère avec lequel il faut remplir la variable.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
VariableARemplir	Sans type	Variable qu'il faut remplir par le caractère donné par ValeurDeRemplissage.

But de la fonction :

Remplir avec le caratère ValeurDeRemplissage les Taille premiers octets de la variable VariableARemplir.

Il est conseillé d'utiliser la fonction **sizeof**, avec comme argument la variable VariableARemplir, comme paramètre effectif pour le paramètre formel Taille.

Nom de la procédure : **PAttendre**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Secondes	byte	Nombre de secondes d'attente.
Centiemes	byte	Nombre de centièmes de seconde d'attente.
BoutonDroitSourisActif	boolean	Indique si l'événement 'clique du bouton droit de la souris' est pris en compte.

But de la fonction :

Générer une pause de Secondes secondes et Centièmes centièmes de seconde.
Si BoutonDroitSourisActif = TRUE, un événement 'clique du bouton droit de la souris' provoque l'interruption de l'attente.

Si Centiemes est supérieur à 99, la procédure est sans effet.

Nom de la fonction : **FTrajectoire**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
PointOrigine	Ppoint	Pointeur sur une instance du type tpoint.
PointDestination	Ppoint	Pointeur sur une instance du type tpoint.
Pas	integer	Distance entre les points intermédiaires.

Valeur retournée par la fonction :

Type : PCollectionPoints

Signification : Collection de points intermédiaires distants d'environ Pas unités et situés approximativement sur l'axe ' joignant ' le point PointOrigine et le point PointDestination. Si PointOrigine ou PointDestination vaut nil, si Pas <= 0 ou si Pas est tel qu'il n'y a pas de points intermédiaires, la fonction renvoit nil.

3) Fonctions relatives au module Tas.

Nom de la fonction : **FRetournerTypeElements**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Index	byte	Rang d'une constante du type énuméré TETypesElements.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
TypeElements	TETypesElements	Constante du type TETypesElements dont le rang est Index.

Valeur retournée par la fonction :

Type : boolean

Signification : Vaut TRUE si le rang indiqué par le paramètre Index n'est pas hors limite, FALSE dans le cas contraire.

But de la fonction :

Si Index est hors limite c'est-à-dire s'il ne correspond pas de constante du type TETypesElements au rang Index, TypeElement n'est pas modifié sinon le paramètre TypeElement prend comme valeur la constante énumérée du type TETypesElements dont le rang est donné par le paramètre Index.

Nom de la fonction : **FRetournerTailleElements**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Index	byte	Rang d'une constante du type énuméré TETaillesElements.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
Taille	TETaillesElements	Constante du type TETaillesElements dont le rang est Index.

Valeur retournée par la fonction :

Type : boolean

Signification : Vaut TRUE si le rang indiqué par le paramètre Index n'est pas hors limite, FALSE dans le cas contraire.

But de la fonction :

Si Index est hors limite c'est-à-dire s'il ne correspond pas de constante du type TETaillesElements au rang Index, Taille n'est pas modifié sinon le paramètre Taille prend comme valeur la constante énumérée du type TETaillesElements dont le rang est donné par le paramètre Index

Nom de la méthode : **FCreerRegion**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
NomElement	integer	Entier identifiant une des icônes-éléments.

Valeur retournée par la fonction :

Type : hrgn

Signification : Région (au sens Windows) polygonale coïncidant approximativement avec le contour du motif de l'icône-élément identifiée par le paramètre NomElement.
La région créée englobe le motif de l'icône.

B) Notes pour les utilisateurs de la bibliothèque d'objets.

Types

TETypesElements = (ETomate,ECitron,EBalle,ERien,EMultiType)

Ce type énuméré reprend une constante pour chaque type d'éléments (tomate, orange, bonbon, ...) pour lequel trois icônes ont été conçues. Il comprend en plus deux constantes particulières ERien et EMultiType pour lesquelles aucune icône n'a été conçue.

TETaillesElements = (EMax,EMoy,EMin,EMultiTaille)

Chaque type d'éléments (tomate, orange, ...) comporte trois tailles à savoir : taille maximum, taille moyenne, taille minimum. EMultiTaille est une constante indiquant que les éléments d'un ensemble peuvent être de tailles différentes. A chacune des 3 premières constantes doit correspondre une icône représentant l'élément selon la taille indiquée par cette constante. Ces trois icônes doivent obligatoirement avoir des identificateurs consécutifs et le premier identificateur doit être un multiple de 3 (ex : 102,103,104).
Le premier identificateur (le plus petit) sera toujours associé à l'icône représentant l'élément dans sa plus grande taille. Le second identificateur sera associé à l'icône représentant l'élément dans sa taille moyenne et le troisième identificateur (le plus grand) étant associé à l'icône représentant l'élément dans sa plus petite taille.

Constantes

CTabIndicesElements

Il s'agit d'un tableau reprenant pour chaque constante définie dans le type TETypesElements l'identifiant de la première icône représentant le type d'éléments dans sa taille maximum. En outre, ce tableau comprend également les constantes 0 et -1 pour les constantes énumérées ERien et EMulti.

CNbrElementsDefinis

Indique le nombre de constantes énumérées du type TETypesElements sans prendre en compte les deux constantes ERien et EMulti.

CNbrTailles

Indique le nombre de constantes énumérées du type TETaillesElements sans prendre en compte la constante EMultiTaille.

CTabRegion+Type d'éléments+Taille

Tableau du type array [1..16] of tpoint. Il s'agit donc d'un tableau de points dont les champs (coordonnées) x et y sont >= 0 et <= 31 et dont la longueur réelle est donnée par la constante CLongTabRegion+Type d'éléments+Catégorie.
Si pour tout i : 1 .. (CLongTabRegion+Type d'éléments+Taille)-1, on joint le point i au point i +1 et si on joint le point CLongTabTabRegion+Typed'objet+Taille (c'est-à-dire le dernier) au premier point, on obtient un polygone qui correspond approximativement à la forme du type d'élément Type d'éléments d'une taille Taille dessinée dans une icône. Le motif de l'icône doit être totalement inclus dans la forme définie par les points du tableau.

13. Spécification des objets du module 'GlobAppl'.

Nom de la fonction : **FRetournerBaseChoixExercices**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Index	byte	Rang d'une constante du type énuméré TEBasesChoixExercices.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
Base	TEBasesChoixExercices	Constante du type TEBasesChoixExercices dont le rang est Index.

Valeur retournée par la fonction :

Type : boolean

Signification : Vaut TRUE si le rang indiqué par le paramètre Index n'est pas hors limite, FALSE dans le cas contraire.

But de la fonction :

Si Index est hors limite c'est-à-dire s'il ne correspond pas de constante du type TEBasesChoixExercices au rang Index, Base n' est pas modifié sinon le paramètre Base prend comme valeur la constante énumérée du type TEBasesChoixExercices dont le rang est donné par le paramètre Index.

14. Spécification des objets du module 'Boite'.

Type objet : TBoiteCommentaire

Champs.

Nom	Type	Signification
Fenetre	pwindow	Pointeur sur l'objet fenêtre qui contient la boîte.
Bord	TRectangle	Définit la zone rectangulaire intérieure à la boîte dans laquelle sera affiché Texte.
Titre	pchar	Chaîne de caractères affichée dans la barre de titre.
EncreTitre	longint	Couleur de l'encre pour le titre.
FondTitre	longint	Couleur de fond sur laquelle sera affiché le titre.
EncreTexte	longint	Couleur de l'encre pour le texte.
Texte	array [0 .. MaxCommentaire] of char.	Chaîne de caractères à afficher dans la boîte.
FondBoite	longint	Couleur de fond sur laquelle le texte sera affiché.
Plume	hpen	Plume pour tracer les traits de la boîte.
Effaceur	hpen	Plume pour effacer le contenu de la boîte.
BrosseFond	hbrush	Brosse pour effacer l'intérieur de la boîte.
BrosseTitre	hbrush	Brosse qui peint la barre de titre.
Peinte	boolean	Indique si la boîte est affichée.

X1,Y1 : integer Coordonnées du coin supérieur gauche de la boîte.
X2,Y2 : integer Coordonnées du coin inférieur droit de la boîte.

Ces champs sont hérités du type objet TRectangle. Bien que n'étant pas propres au type spécifié, nous les mentionnons car leur signification ne découle pas directement du type objet ancêtre.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Destructeur : **Done**

But de la méthode : détruire les variables dynamiques qui dépendent de l'objet.

Nom de la méthode : **MPAssignerDuTexte**

Champs modifiés : Texte → UnTexte

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnTexte	pchar	Chaîne de caractères qui est affectée au champ Texte.

But de la méthode :

Affecter au champ *Texte* la valeur du paramètre UnTexte.

Nom de la méthode : **MPAfficherTitre**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.

But de la méthode :

Afficher la barre de titre et son texte *Titre*. Si *Peinte* vaut FALSE, la méthode est sans effet.

Nom de la méthode : **MPAfficherContenu**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.

But de la méthode :

Afficher le champ *Texte* dans la boîte. Si *Peinte* vaut FALSE, la méthode est sans effet.

Nom de la méthode : **MPAfficherBoite**

Champs modifiés : Peinte → TRUE

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.

But de la méthode :

Afficher la boîte (contour, titre et contenu) et affecter au champ *Peinte* la valeur TRUE.

Nom de la méthode : **MPEffacerContenu**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.

But de la méthode :

Effacer le contenu (zone texte) de la boîte. Si la boîte n'est pas affichée, la méthode est sans effet.

Nom de la méthode : **MPEffacerBoite**

Champs modifiés : Peinte → FALSE

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.
CouleurFond	longint	Couleur de fond sur laquelle la boîte a été affichée.

But de la méthode :

Effacer la boîte en utilisant la couleur CouleurFond. Le champ *Peinte* prend la valeur FALSE.

Nom de la méthode : **MPObtenirDC**

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.

But de la méthode :

Obtenir un contexte d'affichage pour la fenêtre *Fenetre*. UnDC est le handle du contexte d'affichage obtenu.

Nom de la méthode : **MPLacherDC**

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.

But de la méthode :

Libérer le contexte d'affichage UnDC et mettre UnDC à 0. Si, au moment de l'appel de la méthode, UnDC vaut 0, la méthode est sans effet.

15. Spécification des objets du module 'Utilisateur'.

Type objet : TResultatsComparaisons

Ce type objet est spécifique à l'application développée.

Champs.

Nom	Type	Signification
CoefficientPonderation	real	Coefficient de pondération[4].
TableauTauxEchecsGlobaux	TTableauTauxEchecs	Tableau dont les éléments sont du type TTauxEchecs.
TableauTauxEchecsDerniers	TTableauTauxEchecs	Tableau dont les éléments sont du type TTauxEchecs.
TableauTauxEchecsPonderes	TTableauTauxEchecs	Tableau dont les éléments sont du type TTauxEchecs.
TableauTauxEchecsUtilisateur	TTableauTauxEchecs	Tableau dont les éléments sont du type TTauxEchecs.

Remarques :

TTauxEchecs = record

NbrEchecs : integer (* Indique le nombre d'échecs *)
NbrTotal : integer (* Indique le nombre d'épreuves réalisées *)
Taux : real (* Indique le taux d'échecs (=NbrEchecs/NbrTotal) *)

Pour ce qui est du tableau TableauTauxEchecsGlobaux[i], on a :

-NbrEchecs qui reprend le nombre de mauvaises réponses données pour la caractéristique/manipulation i depuis le début de la session ou depuis une session ultérieure dans le cas où les résultats sont sauvegardés.
-NbrTotal qui reprend le nombre total de réponses données pour la caractéristique/manipulation i depuis le début de la session ou depuis une session ultérieure dans le cas où les résultats sont sauvegardés.
-Taux qui est le rapport entre ces deux données.

Pour ce qui est du tableau TableauTauxEchecsDerniers[i], on a :

-NbrEchecs qui reprend le nombre de mauvaises réponses données pour la caractéristique/manipulation i sur les NbrTotal dernières réponses pour la caractéristique/manipulation i.

[4] Le coéfficient de pondération (α) est tel que TableauTauxEchecsPondérés[i] = TableauTauxEchecsGlobaux[i] X (1-α) + TableauTauxEchecsDerniers X (α).

-NbrTotal qui est un entier qui indique le nombre de dernières réponses dont on tient compte.
-Taux qui est le rapport entre ces deux données.

Pour ce qui est du tableau TableauTauxEchecsPonderes[i], on a :

-NbrEchecs qui est sans importance.
-NbrTotal qui est sans importance.
-Taux qui est la pondération entre le taux global et le taux pour les X derniers résultats.

Pour ce qui est du tableau TableauTauxEchecsUtilisateur[i], on a :

-NbrEchecs qui est sans importance.
-NbrTotal qui est sans importance.
-Taux qui est une valeur entre 0 et 1 qui est donnée ou modifiée par l'utilisateur.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Constructeur : **Load**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un flux.

But de la méthode :

Construire et charger une instance du type TResultatsComparaisons à partir du flux Flux.

Nom de la méthode : **Store**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un Flux.

But de la méthode :

Enregistrer une instance du type TResultatsComparaisons dans le flux Flux.

Nom de la méthode : **MPMAJTableauResultats**

Champs modifiés : Certaines collections du tableau TableauResultats : (count → [+1]) et éventuellement (limit → +delta)

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Identique	boolean	Indique si les tas proposés lors de l'exercice sont identiques.
Lineaire	boolean	Indique si les tas proposés lors de l'exercice sont linéaires.
Reussite	boolean	Indique si l'exercice proposé a été réussi (TRUE) ou raté (FALSE).
TypeManipulation	TETypesManipulations	Indique le type de manipulation qui a été proposé.

But de la méthode :

Ajouter à la fin de certaines collections une instance du type TUneValeurResultat dont le champ Valeur prendra la valeur du paramètre Reussite. Ce sont les paramètres Lineaire, Identique et TypeManipulation qui déterminent quelles seront les collections modifiées.

Si le nombre d'éléments d'une des collections devant être modifiées est égal au nombre maximum d'éléments autorisés, le premier élément de cette collection sera supprimé et libéré afin de pouvoir ajouter l'élément devant l'être (voir TBoolCollection. MFAjouter).

Nom de la méthode : **MPMAJTableauxTauxEchecs**

Champs modifiés : Les Tableaux TableautauxEchecsXXXX.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Identique	boolean	Indique si les tas proposés lors de l'exercice sont identiques.
Lineaire	boolean	Indique si les tas proposés lors de l'exercice sont linéaires.
Reussite	boolean	Indique si l'exercice proposé a été réussi (TRUE) ou raté (FALSE).
TypeManipulation	TETypesManipulations	Indique le type de manipulation qui a été proposé.

But de la méthode :

Mettre à jour les tableaux TableauTauxEchecsXXXX.

Ce sont les paramètres Lineaire, Identique et TypeManipulation qui déterminent quels seront les éléments des tableaux qui seront modifiés.

Type objet : TUtilisateur

Ce type objet est spécifique à l'application développée.

Champs.

Nom	Type	Signification
Nom	pchar	Nom de l'utilisateur.
Prenom	pchar	Prénom de l'utilisateur.
Resultats	PResultatsComparaisons	Résultats obtenus par l'utilisateur.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Remarques : le champ Resultats pointe sur une nouvelle instance du type TResultatsComparaisons.

Constructeur : **Load**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un flux.

But de la méthode :

Construire et charger une instance du type TUtilisateur à partir du flux Flux.

Nom de la méthode : **Store**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Flux	tstream	Un Flux.

But de la méthode :

Enregistrer une instance du type TUtilisateur dans le flux Flux.

Nom de la méthode : **Done**

But de la méthode : Détruire les variables dynamiques dépendant de l'objet.

16. Module 'Tas'.

A) Spécifications.

1) Procédures.

Nom de la procédure : **PAfficherUnNbrEnConstellations**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.
Nombre	byte	Le nombre à représenter à l'aide des constellations de 1 à 6.
XDepart	integer	Abscisse du coin supérieur droit de la première icône (représentant une constellation) utilisée pour représenter le nombre.
YDepart	integer	Ordonnée du coin supérieur droit de la première icône (représentant une constellation) utilisée pour représenter le nombre.
Horizontal	boolean	Indique si les constellations utilisées pour représenter le nombre sont alignées horizontalement (TRUE) ou verticalement (FALSE).

But de la procédure :

Représenter le nombre Nombre sous forme d'un ensemble de constellations de 1 à 6. Une constellation a une représentation iconique. Si Horizontal vaut TRUE, les constellations sont mises les unes à coté des autres horizontalement. Si Horizontal vaut FALSE, les constellations sont disposées verticalement. La première (la plus à gauche ou la plus haute), a son coin supérieur droit aux coordonnées (XDépart,YDépart).

Nom de la procédure : **PEffacerUnNbrEnConstellations**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.
Nombre	byte	Le nombre, représenté sous forme de constellations, à effacer.
XDepart	integer	Abscisse du coin supérieur droit de la première icône utilisée pour représenter le nombre.
YDepart	integer	Ordonnée du coin supérieur droit de la première icône utilisée pour représenter le nombre.
Horizontal	boolean	Indique si les constellations utilisées pour représenter le nombre sont alignées horizontalement (TRUE) ou verticalement (FALSE).
CoulFond	longint	Couleur de fond sur laquelle les icônes ont été affichées et qui sera utilisée pour les effacer.

But de la procédure :

Effacer un nombre représenté sous forme d'un ensemble de constellations de 1 à 6. Une constellation a une représentation iconique. XDepart et YDepart sont les coordonnées du coin supérieur droit de la première icône. Horizontal indique si les constellations sont disposées horizontalement (TRUE) ou verticalement (FALSE).

2) Types objets

Type objet : TCentreDeRef

Champs.

X,Y : Coordonnées X et Y du centre de référence.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Nom de la méthode : **MPAffecterXY**

Champs Modifiés : X → UnX et Y → UnY

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnX	integer	Nouvelle coordonnée X pour le centre de référence
UnY	integer	Nouvelle coordonnée Y pour le centre de référence

But de la méthode :

Changer les coordonnées du centre de référence. Le champ *X* prend la valeur UnX et le champ *Y* la valeur UnY.

Type Objet : TConstellation

Champs.

Nom	Type	Signification
Nom	integer	Nom de la constellation, prend pour valeur une constante identifiant une des icônes définies représentant une des constellations de 1 à 6.
Visible	boolean	Indique si l'objet visuel (icône) associé à l'objet est dessiné à l'écran.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Nom de la méthode : **MPDessiner**

Champs modifiés : Visible → TRUE

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.

But de la méthode :

Dessiner, dans le contexte d'affichage UnDC, l'icône identifiée par le champ *Nom*. Le coin supérieur droit de l'icône aura les coordonnées *X* et *Y*. Affecter la valeur TRUE à *Visible*.

Nom de la méthode : **MPEffacer**

Champs modifiés : Visible → [FALSE]

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.
CoulFond	longint	Entier long indiquant la couleur de fond sur laquelle l'icône a été affichée et qui sera utilisée pour l'effacer.

But de la méthode :

Effacer l'icône associée à l'objet et affecter la valeur FALSE à *Visible*. Si *Visible* vaut FALSE au moment de l'appel, la méthode est sans effet.

Type objet : TElement

Champs.

Nom	Type	Signification
Selectionne	boolean	Indique si l'élément est sélectionné.
OrdreAffichage	pword	Permet de connaître l'ordre d'affichage des éléments.

Méthodes.

Nom de la méthode : **Init**

But de la méthode : Initialiser l'objet.

Destructeur : **Done**

But de la méthode : détruire les variables dynamiques qui dépendent de l'objet.

Nom de la méthode : **MPDessiner**

Champs Modifiés : (Visible → TRUE)

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage **VALIDE**
UnOrdreAffichage	word	Indique le moment où l'icône associée à l'objet a été affichée. Permet de connaître l'ordre d'affichage des icônes en cas de superposition.

But de la méthode :

Dessiner sur le contexte d'affichage UnDC l'élément visuel (une icône) associé à l'objet. Le coin supérieur gauche de l'icône aura les coordonnées *X* et *Y* .

Mettre le champ Visible à TRUE.

Nom de la méthode : **MPEffacerParRegion**

Champs modifiés : (Visible → [FALSE])

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE
CoulFond	longint	Couleur de fond sur laquelle l'icône associée à l'objet a été affichée et qui sera utilisée pour l'effacer.

But de la méthode :

Effacer l'élément visuel associé à l'objet. Pour ce faire, la méthode appelle la fonction FCreerRegion pour créer une région au sens Windows englobant le motif de l'icône et la remplit de la couleur CoulFond. Le champ Visible prend la valeur FALSE.

Si, au moment de l'appel, Visible vaut FALSE, la méthode est sans effet.

Nom de la méthode : **MPSelectionner**

Champs Modifiés : (Selectionne → Sel)

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Sel	boolean	Valeur qui sera affectée au champ Selectionne.

But de la méthode :

Affecter au champ *Selectionne* la valeur du paramètre Sel.

Nom de la méthode : **MPBarrer**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage **VALIDE**

But de la méthode :

Barrer l'icône associée à l'objet c'est-à-dire tracer une ligne du coin supérieur gauche de l'icône vers le coin inférieur droit de l'icône ainsi qu'une ligne du coin supérieur droit vers le coin inférieur gauche.

Type objet : TCollectionTElement

Méthodes.

Nom de la méthode : **MPPermuter**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
IndexElement1	byte	Indice d'un élément dans la collection.
IndexElement2	byte	Indice d'un élément dans la collection.

But de la méthode :

Vérifier si les indices ne sont pas hors limites c'est-à-dire vérifier si les paramètres IndElement1 et IndElement2 sont >= 0 et < au champ count.

Si les deux indices sont corrects, la méthode permute les deux éléments occupant les positions données par les paramètres.

Nom de la méthode : **MPElementAvecXMaximum**.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Long	integer	Indique la longueur de la portion de la collection sur laquelle la recherche est réalisée.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
Maxi	integer	Indique la plus grande valeur trouvée pour le champ X pour les Long premiers éléments de la collection.
Position	integer	Indique la position du premier élément parmi les Long premiers éléments de la collection dont le champ X vaut Maxi.
Croissant	boolean	Renvoie TRUE si les Long premiers éléments de la collection sont triés par ordre croissant selon le champ X.

But de la méthode :

Vérifier que Long est inférieur ou égal au nombre d'éléments de la collection.

Si c' est le cas :

Rechercher, parmi les Long premiers éléments de la collection, le premier élément ayant la plus grande valeur pour le champ X, affecter à Maxi cette valeur, affecter au paramètre Position, sa position dans la collection et affecter à Croissant la valeur TRUE si ces Long premiers éléments sont triés par ordre croissant selon le champ X.

Sinon :

Affecter au paramètre Position la valeur -1.

Remarques :

- Si la collection contient des pointeurs valant NIL, le paramètre Croissant est renvoyé avec la valeur FALSE.

- Si la collection est vide et que long vaut 0 ou si la collection ne contient que des pointeurs valant NIL, le paramètre Position est renvoyé avec la valeur -2.

Nom de la méthode : **MPElementAvecYMaximum**.

Spécification semblable à la précédente où il convient de remplacer X par Y.

Nom de la méthode : **MPElementAvecOrdreAffichageMaximum**.

Spécification semblable à la précédente où il convient de remplacer X par OrdreAffichage.

Nom de la méthode : **MPTrierSurX**

But de la méthode :

Trier la collection d'éléments selon leur champ X.

Remarque :

Si la collection comporte des pointeurs valant NIL, ceux-ci se retrouvent en tête de la collection.

Nom de la méthode : **MPTrierSurY**

But de la méthode :

Trier la collection d'éléments selon leur champ Y.

Remarque :

- Si la collection comporte des pointeurs valant NIL, ceux-ci se retrouvent en tête de la collection.

Nom de la méthode : **MPTrierSurOrdreAffichage**

But de la méthode :

Trier la collection d'éléments selon le champ OrdreAffichage.

Remarque :

- Si la collection comporte des pointeurs valant NIL, ceux-ci se retrouvent en tête de la collection.

Nom de la méthode : **MFCopier**

Valeur retournée par la fonction :

Type : PCollectionTElement

Signification : Pointe sur une nouvelle collection qui est une copie de la collection appelant la méthode MFCopier.

Type objet : TCollectionTrieeTElement

Méthodes.

Nom de la méthode : **KeyOf**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
item	pointer	Pointeur vers une variable dynamique qui contient la clé de comparaison.

Valeur retournée par la méthode :

Type : pointer

Signification : Pointeur vers la clé utilisée pour la comparaison de deux éléments de la collection.
La collection étant sensée ne contenir que des éléments du type TElement triés selon le champ OrdreAffichage, le pointeur renvoyé par la fonction est égal au champ OrdreAffichage de Item.

Nom de la méthode : **Compare**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Cle1	pointer	Pointeur vers le premier élément de comparaison.
Cle2	pointer	Pointeur vers le deuxième élément de comparaison.

Valeur retournée par la méthode :

Type : integer

Signification : Résultat de la comparaison de Cle1 et Cle2.
Si Cle1 < Cle2, la fonction renvoie -1. Si Cle1 = Cle2, la fonction renvoie 0. Si Cle1 > Cle2, la fonction renvoie 1.

Nom de la méthode : **MPElementAvecXMaximum**.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Long	integer	Indique la longueur de la portion de la collection sur laquelle la recherche est réalisée.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
Maxi	integer	Indique la plus grande valeur trouvée pour le champ X pour les Long premiers éléments de la collection.
Position	integer	Indique la position du premier élément parmi les Long premiers éléments de la collection dont le champ X vaut Maxi.
Croissant	boolean	Renvoie TRUE si les Long premiers éléments de la collection sont triés par ordre croissant selon le champ X.

But de la méthode :

Vérifier que Long est inférieur ou égal à la longueur de la collection.

Si c'est le cas :

Rechercher parmi les Long premiers éléments de la collection le premier élément ayant la plus grande valeur pour le champ X, affecter à Maxi cette valeur, affecter au paramètre Position, la position de l'élément dans la collection et affecter à croissant la valeur TRUE si ces long premiers éléments sont triés par ordre croissant selon le champ X.

Sinon :

Affecter à Position la valeur -1.

Nom de la méthode : **MPElementAvecYMaximum**.

Spécification semblable à la précédente où il convient de remplacer X par Y.

Nom de la méthode : **MPCopier**

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
Destination	PCollectionTElement	Pointe sur une collection qui est une copie de la collection appelant la procédure. La seule différence entre les deux collections est leur type.

But de la méthode :

Créer une nouvelle collection du type TCollectionTElement pointée par Destination et qui est une copie de la collection du type TCollectionTrieeTElement appelant la procédure.

Si, au moment de l'appel, le paramètre Destination est différent de NIL, la collection pointée par le paramètre Destination est détruite.

Nom de la méthode : **MFCopier**

Valeur retournée par la méthode :

Type : PCollectionTrieeTElement

Signification : Pointeur sur une nouvelle collection du type spécifié et qui est une copie de la collection appelant la fonction.

Type objet : TTas

Champs.

Nom	Type	Signification
ZoneAffichage	trect	Rectangle définissant la partie d'un contexte d'affichage sur laquelle les icônes représentant les éléments composant le tas pourront être affichés.
NbrElementsMinimum	integer	Nombre minimum d'éléments que le tas peut contenir.
NbrElementsMaximum	integer	Nombre maximum d'éléments que le tas peut contenir.
NbrElements	integer	Nombre d'éléments composant le tas.
TypeUnique	boolean	Vaut TRUE si le tas ne contient qu'un seul type d'éléments.
TailleUnique	boolean	Vaut TRUE si les éléments du tas sont tous de la même taille.
TypeElements	TETypesElements	Indique le type d'éléments que le tas comprend.
TailleElements	TETaillesElements	Indique quelle est la taille des éléments composant le tas.
Lineaire	boolean	Indique si le tas a une configuration linéaire.
Superposition	boolean	Indique si les éléments composant le tas peuvent se superposer.
TousSelectionnes	boolean	Indique si tous les éléments composant le tas sont sélectionnés.
Composition	PCollectionTElement	Pointeur sur la collection d'éléments composant le tas.
Representation	pcollection	Pointeur sur une collection d'éléments du type TConstellation représentant sous forme de constellations le nombre d'éléments du tas.
CompteurAffichage	word	Compteur pour l'affichage des éléments composant le tas. L'affectation de la valeur courante au champ OrdeAffichage d'un élément lors de son affichage permet en cas de superposition d'éléments de déterminer leur ordre d'affichage.

Remarque :

Les méthodes de l'objet assurent que celui-ci sera toujours dans un état cohérent. Toutefois, si pour une raison ou l'autre, l'objet venait à être dans un état incohérent (Ex : NbrElements <= NbrElementsMinimum, Superposition = FALSE alors que certains éléments se chevauchent,...), l'appel d'une de ses méthodes risque de provoquer un dysfonctionnement.

Méthodes.

Nom de la méthode : **Init**

But de la procédure : Initialiser l'objet.

Remarques : Le champ Composition pointe sur une nouvelle instance du type TCollectionTElement avec ses champs limit et delta initialisés respectivement à 5 et à 2.

Le champ Representation est initialisé à NIL.

Nom de la méthode : **Done**

But de la méthode : Détruire les variables dynamiques dépendant de l'objet.

Nom de la méthode : **MPDefinirZoneAffichage**

Champs modifiés : ZoneAffichage → Rect

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Rect	trect	Zone rectangulaire d'un contexte d'affichage dans laquelle le tas sera dessiné.

But de la méthode :

Affecter au champ *ZoneAffichage* le paramètre Rect.

Nom de la méthode : **MPAjouterUnElement**

Champs Modifiés : NbrElements → [+1] et la collection Composition : (count → [+1]) et éventuellement (limit → +delta).

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnElement	PElement	Pointeur sur un élément destiné à être ajouté au tas.

But de la méthode :

Si l'élément UnElement est conforme aux valeurs des champs *ElementUnique*, *TailleUnique*, *TypeElements* et *TailleElements*, celui-ci est ajouté à la fin de la collection *Compositions* et le champ *NbrElements* est incrémenté de 1.

Si UnElement vaut NIL, un élément vide est inséré à la fin de la collection *Composition* et le champ *NbrElements* est incrémenté de 1.

Si *NbrElements* est supérieur ou égal à *NbrElementsMaximum*, la méthode est sans effet.

Nom de la méthode : **MPAjouterUnElementAUnEndroit**

Champs Modifiés : NbrElements → [+1] et la collection Composition : (count → [+1]) et éventuellement (limit → +delta).

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnElement	PElement	Pointeur sur un élément à ajouter au tas.
UnIndex	integer	Position à laquelle il faut ajouter l'élément UnElement dans la collection Composition

But de la méthode :

Si *NbrElements* ≥ *NbrElementsMaximum* ou UnIndex < 0 ou UnIndex > *NbrElements*, la méthode est sans effet.

Si l'élément UnElement est conforme aux valeurs des champs *ElementUnique*, *TailleUnique*, *TypeElements* et *TailleElements*, celui-ci est ajouté à la position UnIndex dans la collection *Composition* et le champ *NbrElements* est incrémenté de 1.

Si UnElement vaut NIL, un élément vide est inséré à la position UnIndex dans la collection *Composition* et le champ *NbrElements* est incrémenté de 1.

Nom de la méthode : **MFObtenirUnElement**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnIndex	integer	Position, dans la collection Composition, de l'élément renvoyé par la fonction.

Valeur retournée par la méthode :

Type : PElement

Signification : Pointeur sur l'élément occupant la position UnIndex dans la collection *Composition*.

Si UnIndex est < 0 ou ≥ *NbrElements*, la fonction renvoie NIL.

Nom de la méthode : **MPRemplacerUnElement**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnElement	PElement	Pointeur sur l'élément remplaçant.
UnIndex	integer	Position, dans la collection Composition, de l'élément qui va être remplacé.

But de la méthode :

Remplacer l'UnIndexième élément de la collection *Composition* par l'élément UnElement.

Si UnIndex est < 0 ou >= NbrElements, la méthode est sans effet.

Remarque :

Lors du remplacement d'un élément de la collection par un autre, il est conseillé de disposer d'un autre pointeur sur l'élément remplacé sous peine de perdre toute trace de ce dernier et par conséquent d'être dans l'impossibilité de libérer l'espace mémoire qui lui a été alloué.

Nom de la méthode : **MPSupprimerUnElementP**

Champs Modifiés : NbrElements → [-1] et la collection Composition → (count : [-1]).

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnElement	PElement	Pointeur sur un élément destiné à être supprimé du tas.

But de la méthode :

- Vérifier si l'élément UnElement appartient à la collection *Composition*.

- Si c'est le cas et si NbrElements > NbrElementsMinimum, supprimer UnElement de la collection, décaler les éléments suivants et décrémenter le champ *NbrElements* de 1.

Remarque :

Lors de la suppression d'un élément de la collection par un autre, il est conseillé de disposer d'un autre pointeur sur l'élément remplacé sous peine de perdre toute trace de ce dernier et par conséquent d'être dans l'impossibilité de libérer l'espace mémoire qui lui a été alloué.

Nom de la méthode : **MPSupprimerUnElementI**

Champs Modifiés : NbrElements → [-1] et la collection Composition : (count → [-1]).

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnIndex	Entier	Position de l'élément à supprimer dans la collection Composition.

But de la méthode :

Supprimer dans la collection d'éléments *Composition* l'élément occupant la position indiquée par UnIndex, décaler les éléments suivants et décrémenter le champ *NbrElements* de 1. Si la valeur de UnIndex est négative ou supérieure ou égale à la valeur du champ *NbrElements* ou si NbrElements <= NbrElementsMinimum, la procédure est sans effet.

Remarque :

Lors de la suppression d'un élément de la collection par un autre, il est conseillé de disposer d'un autre pointeur sur l'élément remplacé sous peine de perdre toute trace de ce dernier et par conséquent d'être dans l'impossibilité de libérer l'espace mémoire qui lui a été alloué.

Nom de la méthode : **MPDetruireUnElementI**

Champs modifiés : NbrElements → [-1] et la collection Composition : (count → [-1]).

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnIndex	integer	Position de l'élément à supprimer et à détruire dans la collection Composition.

But de la méthode :

Supprimer et détruire l'élément occupant la position UnIndex dans la collection *Composition*, décaler les autres éléments et décrémenter *NbrElements* de 1.

Si UnIndex est < 0 ou $\geq$ *NbrElements* ou si NbrElements $\leq$ NbrElementsMinimum, la méthode est sans effet.

Nom de la méthode : **MPDetruireUnElementP**

Champs modifiés : NbrElements $\rightarrow$ [-1] et la collection Composition $\rightarrow$ (count : [-1]).

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
UnElement	PElement	Pointeur sur l'élément devant être détruit et supprimé de la collection Composition.

But de la méthode :

Supprimer et détruire l'élément UnElement de la collection *Composition*, décaler les autres éléments, décrémenter *NbrElements* de 1 et affecter au paramètre UnElement la valeur NIL.

Si l'élément n'appartient pas à la collection *Composition* ou si NbrElements $\leq$ NbrElementsMinimum, la méthode est sans effet.

Nom de la méthode : **MPDessiner**

Champs modifiés : CompteurAffichage $\rightarrow$ +1*NbrElements

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage **VALIDE**

But de la méthode :

Appeler la méthode TElement.MPDessiner pour chaque élément composant le tas et contenu dans la collection *Composition*. Le champ *CompteurAffichage* est incrémenté de 1 avant l'affichage de chaque icône.

Nom de la méthode : **MPEffacer**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE
CoulFond	longint	Couleur utilisée pour effacer le tas.

But de la méthode :

Effacer le tas du contexte d'affichage UnDC. Pour ce faire, la procédure peint la zone d'affichage du tas avec la couleur CoulFond.

Nom de la méthode : **MPEffacerUnElement**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE
UnElement	PElement	Pointeur sur l'élément à effacer.
CoulFond	longint	Couleur utilisée pour effacer l'élément.

But de la méthode :

Appeler la méthode TElement.MPEffacer pour effacer l'élément UnElement du contexte d'affichage UnDC en utilisant la couleur CoulFond. Si cet élément est superposé sur d'autres éléments, ces derniers restent affichés.

Si l'élément UnElement n'appartient pas à la collection *Composition* ou vaut NIL, la procédure est sans effet.

Nom de la méthode : **MPModifTypeUnique**

Champs Modifiés : TypeUnique → TypeUnic

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
TypeUnic	boolean	Indique si les éléments du tas sont tous de même type.

But de la méthode :

Affecter au champ *TypeUnique* la valeur du paramètre TypeUnic.

Nom de la méthode : **MPModifTailleUnique**

Champs Modifiés : TailleUnique → TailleUnic

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
TailleUnic	boolean	Indique si les éléments du tas sont tous de la même catégorie.

But de la méthode :

Affecter au champ *TailleUnique* la valeur du paramètre formel TailleUnic.

Nom de la méthode : **MPModifConfiguration**

Champs Modifiés : Lineaire → Lin

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Lin	Boolean	Indique si le tas a une configuration linéaire.

But de la méthode :

Affecter au champ *Lineaire* la valeur du paramètre formel Lin.

Nom de la méthode : **MPChangerType**

Champs modifiés : TypeElements → Nouvelle valeur du type TETypesElements et tous les éléments de la collection Composition : Nom → Nouvelle valeur.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Aleatoire	boolean	Indique si le nouveau type d'éléments est choisi aléatoirement.
UnTypeElements	TETypesElements	Indique le nouveau type d'éléments souhaité dans le cas où Aléatoire vaut FALSE.

But de la méthode :

Modifier le nom de chaque élément de la collection *Composition* et affecter au champ *TypeElements* une nouvelle valeur. Le nouveau nom des éléments prend la valeur déterminée par les champs *TailleElements* et *TypeElements* (après affectation de la nouvelle valeur).

Si Aleatoire vaut FALSE, le champ *TypeElements* prend la valeur du paramètre UnTypeElements sinon *TypeElements* prend aléatoirement une des constantes du type TETypesElements.

Nom de la méthode : **MPChangerTaille**

Champs modifiés : TailleElements : Nouvelle valeur du type TETaillesElements et tous les éléments de la collection Composition : Nom → Nouvelle valeur.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Aleatoire	boolean	Indique si la nouvelle taille d'éléments est choisie aléatoirement.
UneTailleElements	TETaillesElements	Indique la nouvelle taille d'éléments souhaitée dans le cas où Aleatoire vaut FALSE.

But de la méthode :

Modifier le nom de chaque élément de la collection et affecter au champ *TailleElements* une nouvelle valeur. Le nouveau nom des éléments prend la valeur

déterminée par le champ *TypeElements* et le champ *TailleElements* (après affectation de la nouvelle valeur).

Si Aleatoire vaut FALSE, le champ *TailleElements* prend la valeur du paramètre UneTailleElements sinon *TailleElements* prend aléatoirement une des constantes du type TETaillesElements.

Nom de la méthode : **MPChangerConfiguration**

Champs modifiés : Lineaire → Not(Lineaire), tous les éléments de la collection Composition : X → Nouvelle valeur ; Y → Nouvelle valeur.

But de la méthode :

Affecter au champ *Lineaire* la valeur inverse de la valeur actuelle.

Modifier les éléments de la collection *Composition* de façon à ce qu'ils forment un tas respectant la nouvelle valeur (contrainte) du champ Lineaire.

Si la collection *Composition* contient des éléments vides, la méthode générera une erreur d'exécution.

Nom de la méthode : **MPEcarterElements**

Champs modifiés : Certains éléments de la collection Composition : X → Nouvelle valeur.

But de la méthode :

Modifier le champ X de certains éléments de la collection Composition de façon à ce qu'ils soient plus écartés.

Nom de la méthode : **MPRapprocherElements**

Champs modifiés : Certains éléments de la collection Composition : X → Nouvelle valeur.

But de la méthode :

Modifier le champ X de certains éléments de la collection Composition de façon à ce qu'ils soient plus rapprochés.

Nom de la méthode : **MPChangerDisposition**

Champs modifiés : Certains éléments de la collection Composition : X → Nouvelle valeur.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Ecarter	boolean	Indique s'il faut écarter ou rapprocher les éléments

But de la méthode :

Si Ecarter vaut TRUE (FALSE), modifier le champ X des éléments de la collection *Composition* de façon à écarter (rapprocher) les éléments du tas.

Nom de la méthode : **MPCreerTas**

Champs modifiés : La collection Composition : (count → NbrElements) et éventuellement (limit → +delta) ou tous les éléments de la collection Composition : X → Nouvelle valeur et Y → Nouvelle valeur.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
ChangerCoord	boolean	Indique s'il s'agit d'une création d'un tas ou d'une modification des coordonnées des éléments composant le tas.

But de la méthode :

Si le paramètre ChangerCoord vaut FALSE, la méthode crée un tas comprenant *NbrElements* éléments respectant les contraintes imposées par les champs *Lineaire*, *Superposition*, *TypeUnique*, *CategotrieUnique et ZoneAffichage*. Si *TypeUnique* (*TailleUnique*) vaut TRUE, les éléments du tas sont tous du même type (taille) à savoir le type indiqué par le champ *TypeElements* (*TailleElements*) sinon un type (taille) est choisi aléatoirement pour chaque élément. Le tas créé ne comprend pas d'éléments vides.

Si le paramètre ChangerCoord vaut TRUE, la méthode change les coordonnées des éléments du tas de façon à ce qu'ils respectent les contraintes *Lineaire* et *Superposition*.

Si le paramètre ChangerCoord vaut TRUE et que le tas comprend des éléments vides, la méthode générera une erreur d'exécution.

Remarques :

Avant l'appel de la méthode, la collection Composition doit avoir été initialisée et ne doit contenir aucun élément.

D'autre part, si NbrElements est supérieur à NbrElementsMaximum, ce qui est incohérent, la collection composition en fin d'exécution de la méthode comprendra NbrElementsMaximum éléments et son champs NbrElements prendra la valeur NbrElementsMaximum.

Nom de la méthode : **MPElementAvecXMaximum**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Long	integer	Indique la longueur de la portion de la collection Composition sur laquelle la recherche est réalisée.

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
Maxi	integer	La plus grande valeur parmi les champs X pour les Long premiers éléments composant le tas.
Position	integer	Indique la position du premier élément, parmi les Long premiers éléments de la collection Composition, dont le champ X vaut Maxi.
Croissant	boolean	Renvoie TRUE si les Long premiers éléments de la collection Composition sont triés par ordre croissant selon le champ X.

But de la méthode :

Vérifier que Long est inférieur ou égal au nombre d'éléments de la collection Composition.

Si c' est le cas :

Rechercher, parmi les Long premiers éléments de la collection, le premier élément ayant la plus grande valeur pour le champ X, affecter à Maxi cette valeur, affecter au paramètre Position, sa position dans la collection et affecter à Croissant la valeur TRUE si ces Long premiers éléments sont triés par ordre croissant selon le champ X.

Sinon :

Affecter au paramètre Position la valeur -1.

Remarques :

- Si la collection contient des pointeurs valant NIL, le paramètre Croissant est renvoyé avec la valeur FALSE.

- Si la collection est vide et que long vaut 0 ou si la collection ne contient que des pointeurs valant NIL, le paramètre Position est renvoyé avec la valeur -2.

Nom de la méthode : **MPElementAvecYMaximum**

Voir méthode précédente où 'X' est remplacé par 'Y'.

Nom de la méthode : **MPElementAvecOrdreAffichageMaximum**

Voir méthode précédente où 'Y' est remplacé par 'OrdreAffichage'.

Nom de la méthode : **MFTrierSurX**

Valeur retournée par la fonction :

Type : PCollectionTElement

Signification : Pointeur sur une copie triée de la collection Composition selon le champ X.

Remarque :

Si la collection comporte des éléments vides, ceux-ci se retrouvent en tête de la collection.

Nom de la méthode : **MFTrierSurY**

Valeur retournée par la fonction :

Type : PCollectionTElement

Signification : Pointeur sur une copie triée de la collection Composition selon le champ Y.

Remarque :

Si la collection comporte des éléments vides, ceux-ci se retrouvent en tête de la collection.

Nom de la méthode : **MFTrierSurOrdreAffichage**

Valeur retournée par la fonction :

Type : PCollectionTElement

Signification : Pointeur sur une copie triée de la collection Composition selon le champ OrdreAffichage.

Remarque :

Si la collection comporte des éléments vides, ceux-ci se retrouvent en tête de la collection.

Nom de la méthode : **MFTousSelectionnes**

Champs Modifiés : TousSelectionnes → Nouvelle valeur

Valeur retournée par la méthode :

Type : boolean

Signification : Vaut TRUE si tous les éléments composant le tas ont été sélectionnés c'est-à-dire si le champ Selectionne de chaque élément du type TElement composant la collection *Composition* vaut TRUE.

Affecter au champ *TousSelectionnes* la valeur renvoyée par la méthode.

Nom de la méthode : **MPChangerCoulElement**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.
UnElement	PElement	Pointeur sur l'élément dont il faut changer la couleur.
Coul	longint	Nouvelle couleur.

But de la méthode :

Changer la couleur de l'élément UnElement dessiné sur le contexte d'affichage UnDC en utilisant la couleur Coul.

Si l'élément UnElement est un élément vide ou n'appartient pas à la collection *Composition*, la méthode est sans effet.

Nom de la méthode : **MPDeplacerUnElement**

Champs modifiés : Un élément de la collection Composition : X → NouveauX, Y → NouveauY et OrdreAffichage → Nouvelle valeur.
CompteurAffichage → Nouvelle valeur.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.
CoulFond	longint	Couleur de fond sur laquelle le tas a été affiché et utilisée pour effacer l'emplacement initial de l'élément.
UnElement	PElement	Pointeur sur l'élément à déplacer.
NouveauX	integer	Nouvelle abscisse de l'élément à déplacer.
NouveauY	integer	Nouvelle ordonnée de l'élément à déplacer.
Direct	boolean	Indique si le déplacement est continu ou pas, un déplacement continu étant un déplacement visible.

But de la méthode :

Déplacer, sur le contexte d'affichage UnDC, l'élément UnElement de sa position initiale vers une nouvelle position donnée par les coordonnées UnNouveauX, UnNouveauY .

Si Direct vaut TRUE, l'élément initial est effacé à l'aide de la couleur Coul puis redessiné avec ses nouvelles coordonnées, *CompteurAffichage* est incrémenté de 1 et le champ OrdreAffichage de l'élément reçoit la valeur *CompteurAffichage* de façon à ce qu'il soit le plus grand parmi les champs OrdreAffichage des éléments composant le tas. Si Direct vaut FALSE, l'élément est effacé puis redessiné un certain nombre de fois sur l'axe joignant sa position initiale et sa position finale de façon à ce que le déplacement soit bien visible. A chaque fois que l'élément est dessiné, *CompteurAffichage* est incrémenté de 1 et le champ OrdreAffichage de l'élément reçoit la valeur *CompteurAffichage* de façon à ce qu'il soit le plus grand parmi les champs OrdreAffichage des éléments composant le tas.

Le fait qu'à un certain moment, l'élément en déplacement en superpose un autre du tas est sans conséquence.

Si l'élément UnElement est un élément vide ou n'appartient pas à la collection *Composition*, la méthode est sans effet.

Nom de la méthode : **MPConstruireRepresentation**

Champs modifiés : La collection Représentation : (Count → NbrElements div 6 + 1) et éventuellement (limit → +delta).

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
XDepart	integer	Abscisse du coin supérieur droit de la première constellation de la collection Representation.
YDepart	integer	Ordonnée du coin supérieur droit de la première constellation de la collection Representation.
Horizontal	boolean	Indique si l'ensemble des constellations sera affiché horizontalement ou verticalement.

But de la méthode :

Construire la collection *Representation* dont les éléments sont du type TConstellation de sorte que lors de l'affichage de ces constellations, elles forment une ligne contiguë horizontale ou verticale selon la valeur du paramètre Horizontal et que le coin supérieur droit de la première constellation ait les coordonnées (XDepart,YDepart).

Remarque : Lors de l'appel de la méthode, Representation doit avoir été initialisé et être vide.

Nom de la méthode : **MPDessinerRepresentation**.

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.

But de la méthode :

Appeler la méthode TConstellation.MPDessiner pour chaque constellation de la collection *Representation.*

Nom de la méthode : **MPEffacerRepresentation**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnDC	hdc	Contexte d'affichage VALIDE.
CoulFond	longint	Couleur utilisée pour effacer les constellations de la collection Representation.

But de la méthode :

Appeler la méthode TConstellation.MPEffacer pour chaque constellation de la collection *Representation*.

Nom de la méthode : **MPViderRepresentation**

Champs modifiés : La collection Representation

But de la méthode :

Supprimer et détruire tous les éléments de la collection *Representation*.

Nom de la méthode : **MFCopier**

Valeur retournée par la fonction :

Type : PTas

Signification : Pointeur sur une copie de l'instance du type TTas appelant la méthode.

But de la méthode :

Renvoyer un pointeur sur une copie de l'instance appelant la méthode.

B) Notes pour les utilisateurs de la bibliothèque d'objets.

Type : TElement

Ce type englobe les caractéristiques et comportements d'un objet appartenant à un tas.

Champs :

Nom	Type	Signification
X	Entier	Abscisse de l'objet.
Y	Entier	Ordonnée de l'objet.
Nom	Entier	Nom de l'objet, prend pour valeur une constante identifiant une des icônes définies représentant un objet.
Selectionne	Booléen	Indique si l'objet est sélectionné.
Visible	Booléen	Indique si l'élément visuel (icône) associé à l'objet est dessiné à l'écran.
OrdreAffichage	Pword	Permet de connaître l'ordre d'affichage des objets.

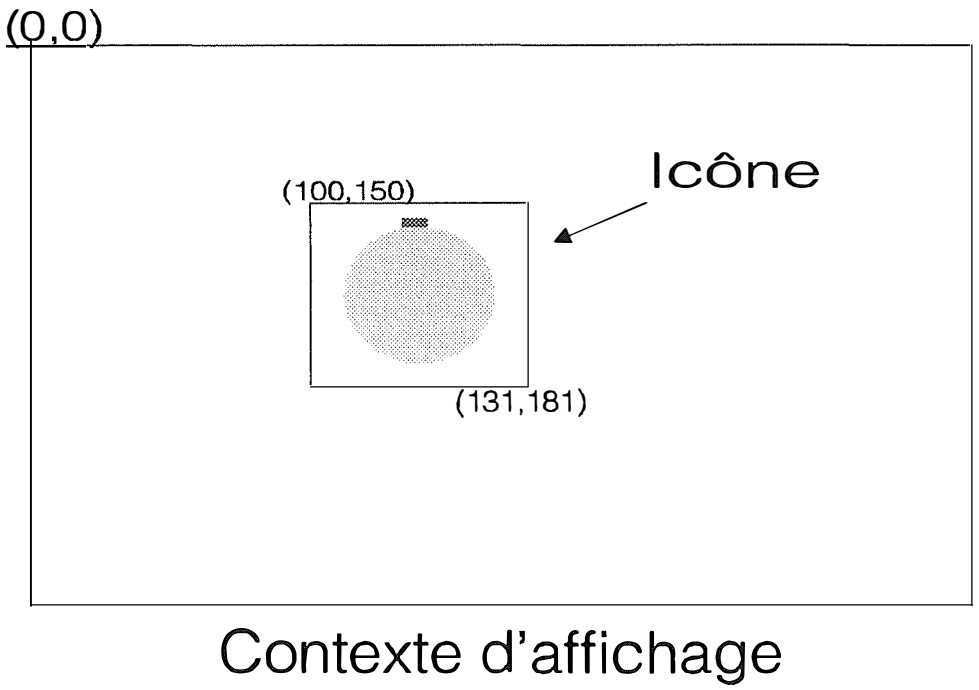
La valeur du champ nom est en fait un entier (ou une constante Pascal entière) qui est l'identifiant d'une icône 32x32 créée à l'aide de **Resource Workshop**. La méthode MPDessine affiche cette icône sur un contexte d'affichage. Les champs X et Y hérités du type objet TCentreDeRef représentent les coordonnées du coin supérieur droit de l'icône dans le contexte d'affichage.

Exemple :

Supposons qu'on ait une instance du type TElement avec les valeurs suivantes pour ses champs X,Y,Nom :

- X : 100
- Y : 150
- Nom : TomateMax (= 105)

Après exécution de la méthode MPDessine pour cette instance et en supposant que le style de projection est le style de projection par défaut c'est-à-dire que l'origine est le coin supérieur gauche du contexte d'affichage, que l'axe des x positifs est pointé vers la gauche et l'axe des y positifs est pointé vers le bas, le résultat sera le suivant:



Type : TCollectionTElement.

Ce type objet est un descendant du type objet *tcollection* fourni par **Borland Pascal 7.0**. Il comprend donc non seulement les méthodes définies mais également les champs et méthodes du type objet *tcollection*. Vous trouverez ces champs et méthodes dans le '*Guide du programmeur*' d' **ObjectWindows** aux pages 375-381.

Le type TCollectionTElement est étroitement lié à deux autres types objets définis dans ce module à savoir les types TElement et TTas. En effet, le type TCollectionTElement est utilisé dans le type TTas afin de représenter la composition d'un tas d'objets.

Une collection du type TCollectionTElement est censée ne contenir **que** des pointeurs vers des objets du type TElement ou des descendants de ce type et éventuellement des pointeurs valant NIL. Toutefois, si une telle collection comprend des descendants du type TElement, la méthode MFCopier ne renverra pas une copie exacte de l'instance appelant la méthode. La copie comprendra uniquement des éléments du type TElement, les éléments d'un type descendant étant convertis en type TElement.

Type : TCollectionTrieeTElement.

Ce type objet est un descendant du type objet *tsortedcollection* (qui est lui même un descendant du type *tcollection*) fourni par **Borland Pascal 7.0**. Il comprend donc non seulement les méthodes définies mais également les champs et méthodes du type objet *tsortedcollection*. Vous trouverez ces champs et méthodes dans le '*Guide du programmeur*' d' **ObjectWindows** aux pages 458-460.

Le type TCollectionTrieeTElement est étroitement lié à deux autres types objets définis dans ce module à savoir les types TElement et TTas. En effet, le type TCollectionTrieeTElement pourrait être utilisé dans le type TTas afin de représenter un tas d'objets.

Une collection du type TCollectionTrieeTElement est censée ne contenir **que** des pointeurs vers des objets du type TElement ou des descendants de ce type. Elle ne peut en aucun cas contenir des pointeurs valant NIL. Toutefois, si une telle collection comprend des descendants du type TElement, la méthode MFCopier ne renverra pas une copie exacte de l'instance appelant la méthode. La copie comprendra uniquement des éléments du type TElement.
De plus, lors de leur insertion dans la collection, les objets sont triés selon la valeur de leur champ OrdreAffichage. Mais si vous modifiez la valeur du champ OrdreAffichage d'un des éléments de la collection, le tri n'est plus assuré être exact.

Type : TTas.

Ce type englobe les caractéristiques et comportements fondamentaux du concept de tas dans le domaine de la conservation ainsi que des fonctionnalités jugées pertinentes et utiles.

Champs :

Nom	Type	Signification
ZoneAffichage	trect	Zone rectangulaire d'un contexte d'affichage dans laquelle les éléments du tas seront affichés.
NbrElementsMinimum	Entier	Nombre minimum d'éléments que le tas peut contenir.
NbrElementsMaximum	Entier	Nombre maximum d'éléments que le tas peut contenir.
NbrElements	Entier	Nombre d'éléments composant le tas.
TypeUnique	Booléen	Egal à vrai si le tas ne contient qu'un seul type d'éléments.
TailleUnique	Booléen	Egal à vrai si les éléments du tas sont tous de la même taille.
TypeElements	TETypesElements	Indique le type d'éléments que le tas comprend.
TailleElements	TETaillesElements	Indique quelle est la taille des objets composant le tas.
Lineaire	Booléen	Indique si le tas a une configuration linéaire.
Superposition	Booléen	Indique si les éléments composant le tas peuvent se superposer.
TousSelectionnes	Booléen	Indique si tous les éléments composant le tas sont sélectionnés.
Composition	PCollectionTElement	Pointeur sur la collection d'éléments composant le tas.
CompteurAffichage	word	Compteur pour l'affichage des éléments composant le tas. Permet en cas de superposition d'éléments de déterminer leur ordre d'affichage.
Representation	pcollection	Pointeur sur une collection d'éléments du type TCollection utilisée pour représenter le nombre d'objets du tas sous forme de constellations.

Comme indiqué précédemment, normalement une collection du type TCollectionTElement peut contenir des pointeurs vides. Cette liberté est également laissée dans le cadre du type objet TTas. En effet, il est possible d'ajouter à un tas, plus précisément à la collection Composition un élément vide bien que cette possibilité n'existe pas dans une situation réelle. Ce choix s'explique par la volonté d'imposer le moins possible de contraintes aux futurs utilisateurs des objets de la

bibliothèque. Cependant, la présence d'un élément vide dans la composition du tas nécessite une grande prudence. En effet, la présence d'éléments vides dans la collection Composition peut provoquer des dysfonctionnements lors de l'utilisation de certaines méthodes du type TTas.

Ainsi, l'exécution des méthodes

- MPCreerTas avec le paramètre ChangerCoord à TRUE
- MPChangerConfiguration
- MPEcarterElements
- MPRapprocherElements
- MPChangerDisposition

en présence d'éléments vides provoquera une erreur d'exécution.

Bien que le type TTas offre de nombreuses méthodes de créations et manipulations de tas, vous pouvez également utiliser les méthodes offertes par le type tcollection pour réaliser des manipulations sur la collection Composition. Cependant l'adéquation entre la collection Composition et les autres champs du type TTas doit être assurée sous peine de dysfonctionnement par la suite.

17. Spécification des objets du module 'Didact'.

Type objet : TFenetreAide

Champs.

Nom	Type	Signification
Message	TMessageAide	Structure indiquant l'identificateur du message actuellement affiché dans la fenêtre et éventuellement d'autres informations.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Destructeur : **Done**

But de la méthode : détruire les variables dynamiques qui dépendent de l'objet.

Nom de la méthode : **CanClose**

Valeur retournée par la fonction :

Type : boolean

Signification : Indique si la fenêtre peut être fermée. La méthode renvoie toujours FALSE et cache la fenêtre.

Nom de la méthode : **Paint**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
paintdc	hdc	Contexte d'affichage.
paintinfo	tpaintstruct	Structure qui contient de l'information pour une application.

But de la méthode :

Dessiner la fenêtre.

Type objet : TFenetreTas

Champs.

Nom	Type	Signification
TableauTas	array [1..CLongMaxTableauTas] of PTas	Tableau de pointeurs pointant sur des instances du type objet TTas représentant des tas d'objets qui pourront être affichés dans la fenêtre.
NbrTas	1..CLongMaxTableauTas	Indique le nombre d'instances du type objet TTas pointées par le tableau TableauTas.
TasCourant	0..CLongMaxTableauTas	Indice du tas courant.
ContexteAffichage	hdc	Contexte d'affichage de la fenêtre.
Crayon	hpen	Outil crayon utilisé pour dessiner dans la fenêtre.
Brosse	hbrush	Outil brosse (pinceau) utilisé pour dessiner dans la fenêtre.
Icones	array[1..CNbrMaxIcones] of TIcone	Tableau reprenant les coordonnées du coin supérieur droit et le handle d'icônes affichées dans la fenêtre.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Remarques :

Les différents pointeurs du tableau TableauTas sont initialisés à NIL.
Crayon est initialisé avec les attributs suivants :

lopnstyle := PS_DOT
lopnwidth := 1
lopncolor := Black

Brosse est initialisé avec les attributs suivants :

lbstyle := PS_SOLID
lbcolor := White

Destructeur : **Done**

But de la méthode : détruire les variables dynamiques qui dépendent de l'objet.

Nom de la méthode : **SetUpWindow**

But de la méthode :

Initialiser l'élément interface associé à l'objet c'est-à-dire la fenêtre. La méthode désactive la fenêtre.

Nom de la méthode : **Paint**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
paintdc	hdc	Contexte d'affichage.
paintinfo	tpaintstruct	Structure qui contient de l'information pour une application.

But de la méthode :

Dessiner la fenêtre.

Nom de la méthode : **GetClassName**

Valeur retournée par la méthode :

Type : pchar

Signification : Nom de la classe de fenêtre

But de la méthode :

Renvoyer le nom de la classe de fenêtre dont la fenêtre fait partie à savoir 'TFenetreTas'.

Nom de la méthode : **GetWindowClass**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
awndclass	twndclass	Structure qui contient les attributs de la classe.

But de la méthode :

Changer un des attributs de la classe 'TFenetreTas' à savoir le curseur. Le curseur d'une fenêtre de la classe ne sera plus le curseur standard fléché mais une main.

Nom de la méthode : **WMLButtonDown**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message Windows WM_LBUTTONDOWN.

But de la méthode :

Répondre à un clique du bouton gauche de la souris. En fait, la méthode vérifie si un objet du tas courant a été cliqué. Si c'est le cas et que celui-ci n'a pas encore été sélectionné (son champ Selectionne vaut FALSE), son champ Selectionne est mis à TRUE et un rectangle est dessiné autour (32x32).

Remarques :

1) Le tas courant ne peut pas être un tas vide (c'est-à-dire TableauTas[TasCourant] <> de NIL).

2) La collection Composition du tas courant ne peut pas être une collection vide (c'est-à-dire TableauTas[TasCourant]^.Composition <> de NIL).

3) Le champs Parent doit être différent de NIL et pointé sur une instance du type TFenetrePrincipale dont le champ Gerant doit être différent de NIL.

Nom de la méthode : **MPChangerTitre**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnTitre	pchar	Titre de la fenêtre.

But de la méthode :

Changer le titre de la fenêtre.

Type objet : TFenetrePrincipale

Champs.

Nom	Type	Signification
Gerant	PGerantDidact	Pointeur vers un objet chargé de gérer l'évolution du logiciel .
NomFichier	array [1..FSPATHNAME] of char	Nom d'un fichier, chemin compris.
FenetreTas	PFenetreTas	Pointeur vers une fenêtre destinée à contenir les différents tas.
FenetreAide	PFenetreAide	Pointeur vers une fenêtre dont le contenu est une aide contextuelle.
Boutons	pcollection	Ensemble de boutons.
TypeBoutons	TEBoutons	Indique quels sont les boutons qui sont contenus dans le champ Boutons.
Bouton1	pbutton	Un bouton.
BoutonAide	pbutton	Un bouton.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Destructeur : **Done**

But de la méthode : Détruire les variables dynamiques qui dépendent de l'objet.

Nom de la méthode : **SetUpWindow**

But de la méthode :

Initialiser l'élément interface associé à l'objet c'est-à-dire la fenêtre.

Nom de la méthode : **CanClose**

Valeur retournée par la méthode :

Type : boolean

Signification : Indique si la fenêtre peut être fermée. Renvoie toujours TRUE. Aucun test n'est réalisé pour savoir si les fenêtres enfants peuvent être fermées.

Nom de la méthode : **Paint**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
paintdc	hdc	Contexte d'affichage.
paintinfo	tpaintstruct	Structure qui contient de l'information pour l'application.

But de la méthode :

Dessiner la fenêtre.

Nom de la méthode : **MFConstruireBouton**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnId	integer	Identificateur du contrôle.
UnTexte	pchar	Texte du bouton.
UnX	integer	Coordonnée x du coin supérieur gauche du bouton.
UnY	integer	Coordonnée y du coin supérieur gauche.
UneLargeur	integer	Largeur du bouton.
UneHauteur	integer	Hauteur du bouton.
Defaut	boolean	Indique si le bouton a l'apparence du bouton par défaut.

Valeur retournée par la méthode :

Type : pbutton

Signification : Pointeur sur une nouvelle instance du type objet tbutton dont la valeur des champs est donnée par les paramètres.

Nom de la méthode : **MPAjouterBouton**

Champs modifiés : La collection Boutons : (count → +1) et éventuellement (limit → +delta).

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnBouton	pbutton	Bouton à ajouter à la collection.

But de la méthode :

Ajouter à la collection *Boutons* le bouton UnBouton.

Nom de la méthode : **MPDetruireBouton**

Champs modifiés : (count → [-1])

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnIndex	integer	Rang du bouton à supprimer de la collection.

But de la méthode :

Détruire dans la collection *Boutons* le bouton occupant la place UnIndex.
Si UnIndex est négatif ou supérieur ou égal au nombre de boutons de la collection, la méthode est sans effet.

Nom de la méthode : **MPAfficherBouton**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnBouton	pbutton	Bouton à afficher.

But de la méthode :

Afficher le bouton UnBouton. Si celui-ci n'a pas encore été créé, la méthode le fait.
Si UnBouton vaut NIL, la méthode est sans effet.

Nom de la méthode : **MPEffacerBouton**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnBouton	pbutton	Bouton à effacer.

But de la méthode :

Cacher le bouton UnBouton. Si UnBouton vaut NIL, la méthode est sans effet.

Nom de la méthode : **MPAffichertousLesBoutonsDeLaCollection**.

But de la méthode :

Afficher tous les boutons de la collection *Boutons*. Le fait que la collection comprenne des pointeurs valant NIL est sans importance.

Nom de la méthode : **MPEffacerTousLesBoutonsDeLaCollection**.

But de la méthode :

Effacer tous les boutons de la collection *Boutons*. Le fait que la collection comprenne des pointeurs valant NIL est sans importance.

Nom de la méthode : **MPAfficherMessage**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
UnMessage	pchar	Message à afficher.
UnTitre	pchar	Titre de la boîte à message.

But de la méthode :

Afficher le message UnMessage dans une boîte à message contenant un bouton OK et dont le titre est la valeur du paramètre UnTitre.

Nom de la méthode : **MPBoutonXXXX**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Appeler la méthode TraitementBouton du gérant pointé par le champ *Gerant* afin de répondre correctement à l'événement (le click du bouton XXXX).

Nom de la méthode : **MPNouveau**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Afficher une boîte de dialogue permettant à l'utilisateur d'entrer son nom et son prénom. Si le champ Utilisateur du gérant *Gerant* est différent de NIL, l'objet pointé par Utilisateur est détruit et une nouvelle instance du type objet TUtilisateur est créée avec comme valeur pour ses champs Nom et Prenom les valeurs entrées dans la boîte de dialogue. Le champ Resultats de la nouvelle instance du type TUtilisateur pointe sur une nouvelle instance du type TResultatsComparaison avec les valeurs 9 et CoefficientPonderationParDefaut pour respectivement les paramètres LongTabResultats et UnCoefficientPonderation de son constructeur Init.

Nom de la méthode : **MPOuvrir**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Créer et charger une instance du type objet TUtilisateur. La nouvelle instance est pointée par le champ Utilisateur du gérant. Si Utilisateur est différent de NIL au moment de l'appel de la méthode, l'instance pointée par Utilisateur est détruite. Si le nombre d'éléments de la $i^{ème}$ collection du tableau TableauResultats de la nouvelle instance du type TUtilisateur est supérieur au champ NbrDerniersResultats du gérant, les éléments les plus anciens (c'est-à-dire les premiers dans la collection) sont supprimés de sorte que le nombre d'éléments de la collection soit égal au champ NbrDerniersResultats du gérant. Les items du menu Resultats sont activés ou grisés selon que le nombre d'éléments de chaque collection du tableau TableauResultats est égal ou différent de NbrDerniersResultats.

La boîte de dialogue standard fichier d'ObjectWindows est utilisée pour sélectionner un fichier.

Nom de la méthode : **MPSauver**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Sauvegarder dans un fichier l'instance du type TUtilisateur pointée par le champ Utilisateur du gérant.

La boîte de dialogue standard fichier **d'ObjectWindows** est utilisée pour sélectionner un fichier.

Nom de la méthode : **MPCommencerExercice**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Appeler les méthodes du gérant permettant l'initialisation et le lancement d'un exercice. L'exercice précédant cette commande doit être préalablement terminé sinon la méthode est sans effet.

Nom de la méthode : **MPTerminerExercice**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Appeler la méthode du gérant permettant de mettre fin à un exercice.

Nom de la méthode : **MPChangerTauxUtilisateur**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Appeler la méthode MPChangerParametres du gérant. Les champs du paramètre du type TMessageParametre ont les valeurs suivantes :

Identificateur = Id_Utilisateur et Pointeur = NIL

Nom de la méthode : **MPAfficherTauxGlobaux**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Exécuter un dialogue permettant à l'utilisateur de connaître ses résultats globaux.

Nom de la méthode : **MPAfficherTauxDerniers**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Exécuter un dialogue permettant à l'utilisateur de connaître ses résultats pour ses X derniers résultats, X étant un paramètre du gérant.

Nom de la méthode : **MPAfficherTauxPonderes**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Exécuter un dialogue permettant à l'utilisateur de connaître ses résultats pondérés.

Nom de la méthode : **MPOptionsExercices**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Appeler la méthode MPChangerParametres du gérant. Les champs du paramètre du type TMessageParametre ont les valeurs suivantes :

Identificateur = Id_Exercices et Pointeur = NIL

Nom de la méthode : **MPAutresTaux**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
message	tmessage	Structure qui contient l'information d'un message.

But de la méthode :

Appeler la méthode MPChangerParamètres du gérant. Les champs du paramètre du type TMessageParametre ont les valeurs suivantes :

Identificateur = Id_Autres_Taux et Pointeur = NIL

Type objet : TGerantVide

Cet objet est un objet générique dont seuls le constructeur Init, le destructeur Done et la méthode Commentaire sont effectifs.

Champs.

Nom	Type	Signification
Fenetre	PFenetrePrincipale	Pointeur sur la fenêtre du logiciel.
Termine	boolean	Vaut TRUE si l'exercice courant est terminé.
Commentaire	PBoiteCommentaire	Objet boîte de texte permettant de faire des commentaires, donner des instructions, ...

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Destructeur : **Done**

But de la méthode : détruire les variables dynamiques qui dépendent de l'objet.

Nom de la méthode : **MPCommentaire**

Paramètres formels produits ou modifiés :

Nom du paramètre	*Type*	*Signification*
UnTitre	pchar	Titre de la boîte à commentaires.
UnId	integer	Identificateur, dans la table de chaînes de caractères, du texte à afficher dans la boîte à commentaires.

But de la méthode :

Afficher du texte dans une boîte du type objet TBoite dont le titre est UnTitre et le texte la chaîne de caractères identifiée par le paramètre UnId dans la table de chaînes de caractères.

Si UnId vaut 0 et que la boîte à commentaires est déjà dessinée, la méthode efface le contenu de la boîte.

Si *Commentaire* vaut NIL, la méthode est sans effet.

Toutes les méthodes qui suivent sont sans effet.

Nom de la méthode : **MPDeterminerExercice**

But de la méthode :

Déterminer le prochain type d'exercice/manipulation.

Nom de la méthode : **MPInitExercice**

But de la méthode :

Initialiser un exercice.

Nom de la méthode : **MPLancerExercice**

But de la méthode :

Lancer un exercice. Normalement, l'exercice doit préalablement avoir été initialisé.

Nom de la méthode : **MPTraitementBouton**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
Numero	byte	Identificateur du bouton à cliquer.

But de la méthode :

Répondre de manière appropriée au bouton cliqué.

Nom de la méthode : **MPTerminerUnExercice**

But de la méthode :

Mettre fin à l'exercice en cours.

Nom de la fonction : **MPChangerParametres**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
MessageParametre	TMessageParametre	Structure qui contient de l'information utile lors de la modification d'un ou plusieurs paramètres.

But de la fonction :

Changer un ou plusieurs paramètres du logiciel.

Nom de la fonction : **MPAide**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
MessageAide	TMessageAide	Identificateur indiquant le contexte dans lequel se trouve l'utilisateur.

But de la fonction :

Fournir l'aide appropriée selon le contexte dans lequel se trouve l'utilisateur.

Type objet : TGerantDidact

Cet objet est responsable du bon déroulement de l'application. C'est à lui de réagir en fonction des événements, des actions de l'enfant.

Nous vous renvoyons au chapitre 'Guide de l'utilisateur' pour connaître la logique de fonctionnement du logiciel.

Contrairement aux autres types d'objets, la spécification des méthodes de cet objet ne comprend pas l'élément 'Champs modifiés'. Nous vous renvoyons au listing pour connaître les effets d'une méthode sur les champs de l'objet. La raison de cette absence est que ce type objet est un type objet spécifique à l'application développée. Cet objet ne sera normalement jamais utilisé dans une autre application, il peut juste servir d'exemple aux futurs concepteurs d'applications.

Champs.

Nom	Type	Signification
MenusItemsGrisesActifs	boolean	Indique si les items du menu 'Résultats' sont grisés ou non.
BaseChoixExercices	TEBasesChoixExercices	Indique quelles données (type de taux) le logiciel doit prendre en compte pour choisir le type d'exercices/manipulations qu'il va proposer à l'enfant.
NbrElementsMin	byte	Indique le nombre minimum d'éléments qu'un tas peut contenir.
NbrElementsMax	byte	Indique le nombre maximum d'éléments qu'un tas peut contenir.
HistoriqueActif	boolean	Indique si l'historique est actif.
ManipulationsVisibles	boolean	Indique si une manipulation est visible ou instantanée.
AfficherMessagesReussites	boolean	Indique s'il faut afficher un message en cas de réussite.
NbrDerniersResultats	byte	Nombre de dernières réponses que l'on prend en compte pour calculer le taux d'échecs sur les X (=NbrderniersResultats) dernières réponses.
TasIdentiques	boolean	Indique si les tas sont identiques.
TasLineaires	boolean	Indique si les tas sont linéaires.
TypeManipulation	TETypesManipulations	Indique le type de manipulation réalisé.
NbrManipulations	byte	Nombre de manipulations qui seront faites au cours d'un exercice avant de passer au suivant.
Erreur	boolean	Indique si l'enfant s'est trompé.
Stade	word	Indique le stade dans lequel l'enfant se trouve.
Utilisateur	PUtilisateur	Pointeur sur un objet qui reprend diverses informations sur l'utilisateur.
Historique	PHistoriqueOctets	Permet la mémorisation d'un certain nombre d'octets.

Méthodes.

Constructeur : **Init**

But de la méthode : Initialiser l'objet.

Destructeur : **Done**

But de la méthode : détruire les variables dynamiques qui dépendent de l'objet.

Nom de la méthode : **MPDeterminerExercice**

But de la méthode :

Déterminer les caractéristiques de l'exercice proposé.

Nom de la méthode : **MPInitExercice**

But de la méthode :

Initialiser les différents objets nécessaires au déroulement d'un exercice.

Nom de la méthode : **MPLancerExercice**

But de la méthode :

Lancer l'exercice. Il faut que les différents objets nécessaires au bon déroulement d'un exercice aient préalablement été initialisés avec la méthode MPInitExercice.

Nom de la méthode : **MPTraitementBouton**

But de la méthode :

Répondre correctement à l'événement 'Click d'un bouton' selon le bouton cliqué et le contexte dans lequel on se trouve.

Nom de la méthode : **MPDeterminerManipulation**

But de la méthode :

Déterminer quelle sera la prochaine manipulation et sur quel tas cette dernière aura lieu.

Nom de la méthode : **MPRéaliserManipulation**

But de la méthode :

Réaliser la manipulation préalablement déterminée.

Nom de la méthode : **MPTerminerExercice**

But de la méthode :

Mettre fin, éventuellement prématurément, à l'exercice en cours. Les différents objets relatifs à l'exercice sont détruits.

Nom de la méthode : **MPChangerParametres**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
MessageParametre	TMessageParametre	Structure contenant de l'information pour un changement de paramètre(s).

But de la méthode :

Changer un ou plusieurs paramètres. Les paramètres modifiés dépendent du paramètre MessageParametre. La modification peut être interne (logicielle) ou externe (utilisateur via un dialogue).

Nom de la méthode : **MPAide**

Paramètres formels non modifiés :

Nom du paramètre	*Type*	*Signification*
MessageAide	TMessageAide	Indique le stade dans lequel l'utilisateur se trouve afin d'afficher le message d'aide approprié.

But de la méthode :

Afficher le message d'aide approprié selon le contexte dans lequel l'utilisateur se trouve.

Partie 4 : GUIDE DE L'UTILISATEUR.

18. Utilisation du logiciel.

A) Description générale.

L'application propose à un enfant des exercices lui permettant de comprendre la notion de conservation de quantités discontinues ainsi que les quantificateurs 'plus, moins, autant, différent'.

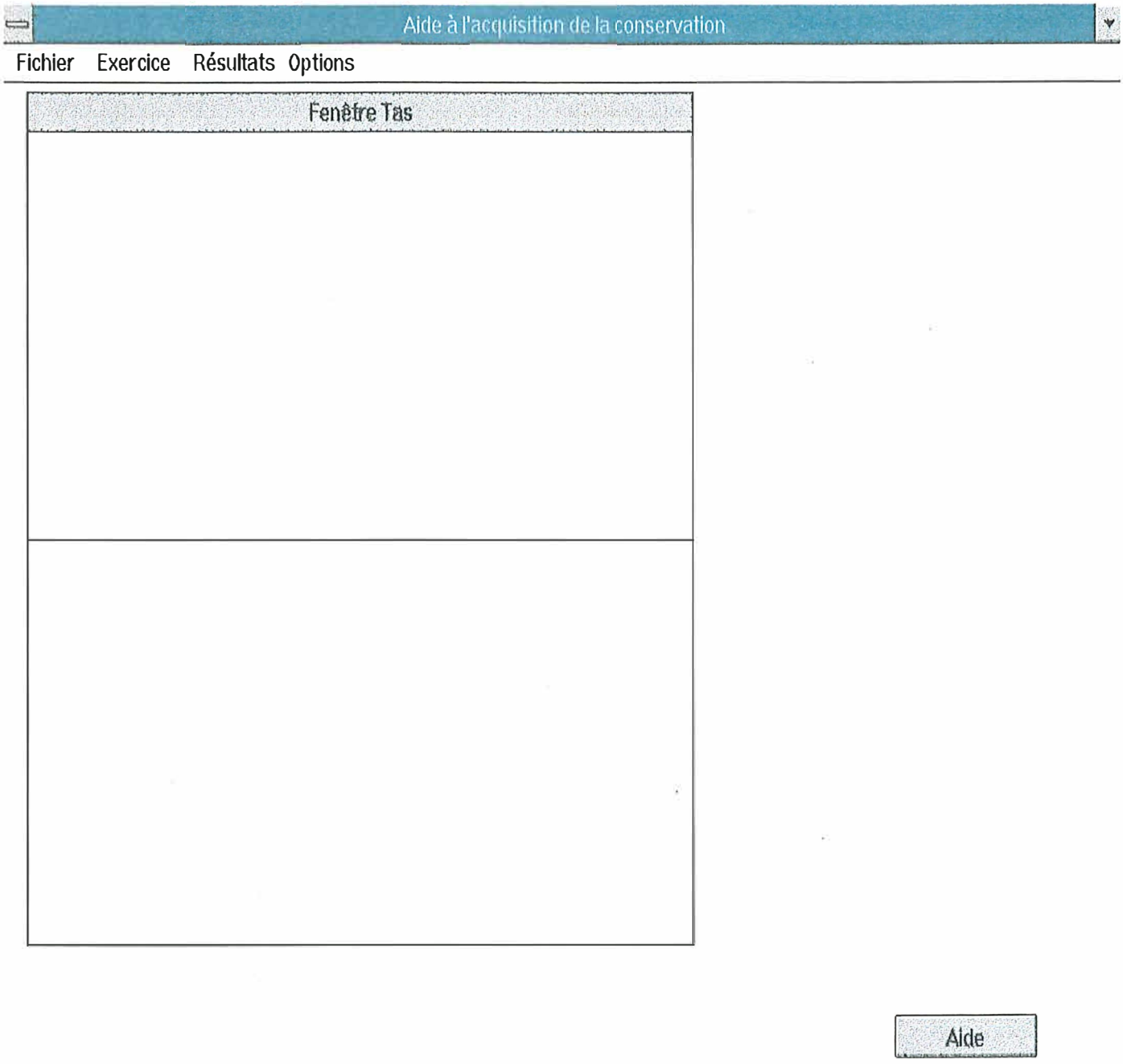
B) Description de l'écran.

1) Dans son état initial, l'écran comprend les éléments suivants :

- La fenêtre principale de l'application : 'Aide à l'acquisition de la conservation'.

- La fenêtre 'Fenêtre tas'.

- Le bouton-poussoir 'Aide'.


Aide à l'acquisition de la conservation
Fichier Exercice Résultats Options
Fenêtre Tas
Aide

2) Lors de la résolution d'un exercice, l'écran comprend les éléments suivants:

a) Eléments permanents :

- Le fenêtre principale de l'application.

- La fenêtre 'Fenêtre tas'.

    Usage : Afficher deux tas (ensembles) d'objets distincts c'est-à-dire dont les objets ne sont pas mélangés et éventuellement la représentation de leur quantité sous forme de constellation de 1 à 6.

- Le bouton d'aide.

    Usage : Fournir une aide contextuelle lorsque l'enfant clique ce bouton.

- La boîte de texte 'INSTRUCTIONS/REMARQUES'

    Usage : .Indiquer à l'enfant ce qu'il doit faire. Le titre de la boîte est alors 'INSTRUCTIONS'.

    . Indiquer un message ou une explication à l'enfant. Le titre de la boîte est alors 'REMARQUES'.

b) Eléments non permanents :

- Un ensemble de 2 ou trois boutons situés en dessous de la fenêtre 'Fenêtre tas'.

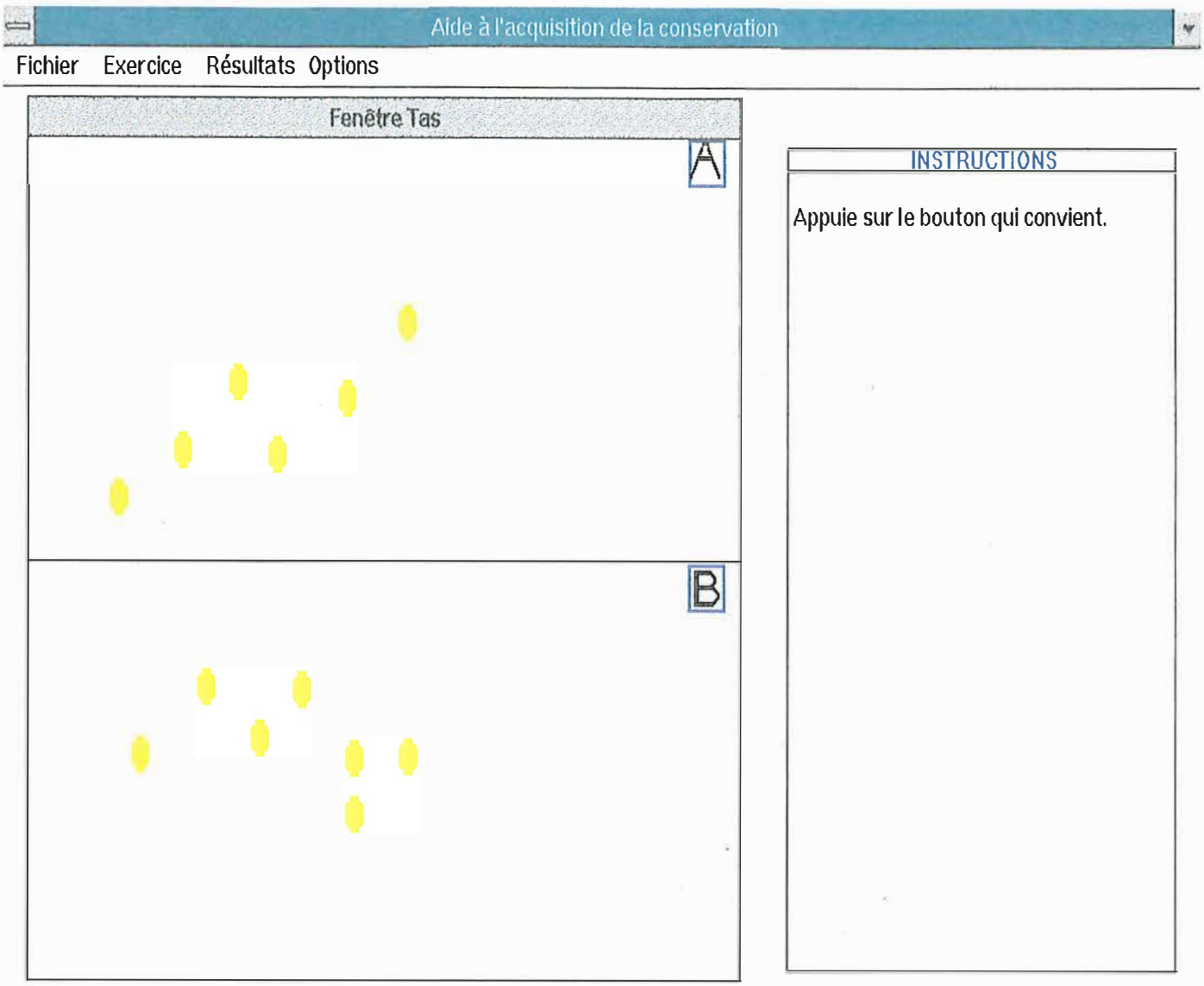
    Usage : L'enfant doit cliquer sur l'un de ces boutons pour donner une réponse.

- Un bouton 'Fini de compter' situé en dessous de la fenêtre 'Fenêtre tas'.

    Usage : Permettre à l'enfant d'indiquer qu'il a fini de compter les objets d'un tas.

- Une fenêtre 'Aide contextuelle'.

    Usage : Fournir une aide à l'enfant en fonction du contexte dans lequel il se trouve.


Aide à l'acquisition de la conservation
Fichier Exercice Résultats Options
Fenêtre Tas
A
B
INSTRUCTIONS
Appuie sur le bouton qui convient.
Autant
Différent
Aide

C) Déroulement d'un exercice.

La description du déroulement d'un exercice va être présentée sous la forme d'un graphe Etat/Evénement. Ce style de graphe reprend un certain nombre d'états. Le passage d'un état à un autre est provoqué par un ou plusieurs événements.

Afin de ne pas surcharger cette description, tout se passe comme si seul le menu 'Exercice' était présent. En fait, on ne prend pas en compte le fait qu'à tout moment , l'utilisateur puisse choisir un item de la barre de menu ainsi que la possibilité pour l'utilisateur de cliquer à tout moment sur le bouton d'aide. En fait, ces deux événements sont sans conséquence pour le déroulement d'un exercice. On ne prend pas non plus en compte la possibilité de mettre fin à un exercice en choisissant l'item 'Exercice' de la barre de menu puis l'item 'Terminer' du menu déroulant.

1) Définition des états possibles.

Etat 1. : L'application dans son état initial.

Description : Voir section précédente.

Etat 2. : Réponse attendue.

Description : Deux ensembles d'objets dénommés A et B sont affichés dans la fenêtre 'Fenêtre tas' ainsi qu'un ensemble de boutons en dessous de cette même fenêtre.

Etat 3. : L'enfant n'a pas encore commencé à compter les objets du tas courant.

Description : Deux tas sont affichés dans la fenêtre 'Fenêtre tas' ainsi que le bouton 'Fini de compter' en dessous de cette même fenêtre.

Etat 4. : L'enfant a compté une partie des objets du tas courant.

Description : Identique à celle de l'état précédent mais un certain nombre d'objets du tas courant sont entourés d'un carré tracé en pointillé.

Etat 5. : L'enfant a compté tous les objets du tas courant.

Description : Identique à la précédente excepté que tous les objets du tas courant sont entourés d'un carré tracé en pointillé.

Etat 6. : Les tas sont mis en correspondance terme à terme.

Description : Si les deux ensembles ont une configuration linéaire, les objets sont reliés par une droite sinon, la correspondance a lieu en utilisant un jeu de couleurs.

2) Description des événements possibles :

a) De la part de l'enfant.

Event 1. : Choisir l'item 'Exercice' de la barre de menu puis choisir dans le menu l'item 'Commencer'.

Event 2. : Répondre en cliquant sur un des boutons situés en dessous de la fenêtre 'Fenêtre tas'.

Event 3. Compter un objet du tas.

Event 4. : Cliquer sur le bouton 'Fini de compter'.

Event 5. : Cliquer à un mauvais endroit.

b) De la part du logiciel.

Event 6. : Réaliser une manipulation sur un des 2 tas d'objets. Les manipulations possibles sont les suivantes :

Changement du type d'objets, de la taille des objets, de la disposition (écarter/ rapprocher les objets), de la configuration (linéaire/quelconque), suppression d'un objet.

Event 7. : Passer au tas suivant.

Event 8. : Mettre en correspondance terme à terme.

Les objets des deux tas sont mis en correspondance terme à terme. Si les deux tas sont linéaires, les objets des deux tas sont reliés. Si la configuration de l'un d'entre eux n'est pas linéaire, la correspondance terme à terme se fait à l'aide d'un jeu de couleurs.

Event 9. : Envoyer un message dans la boîte de texte.

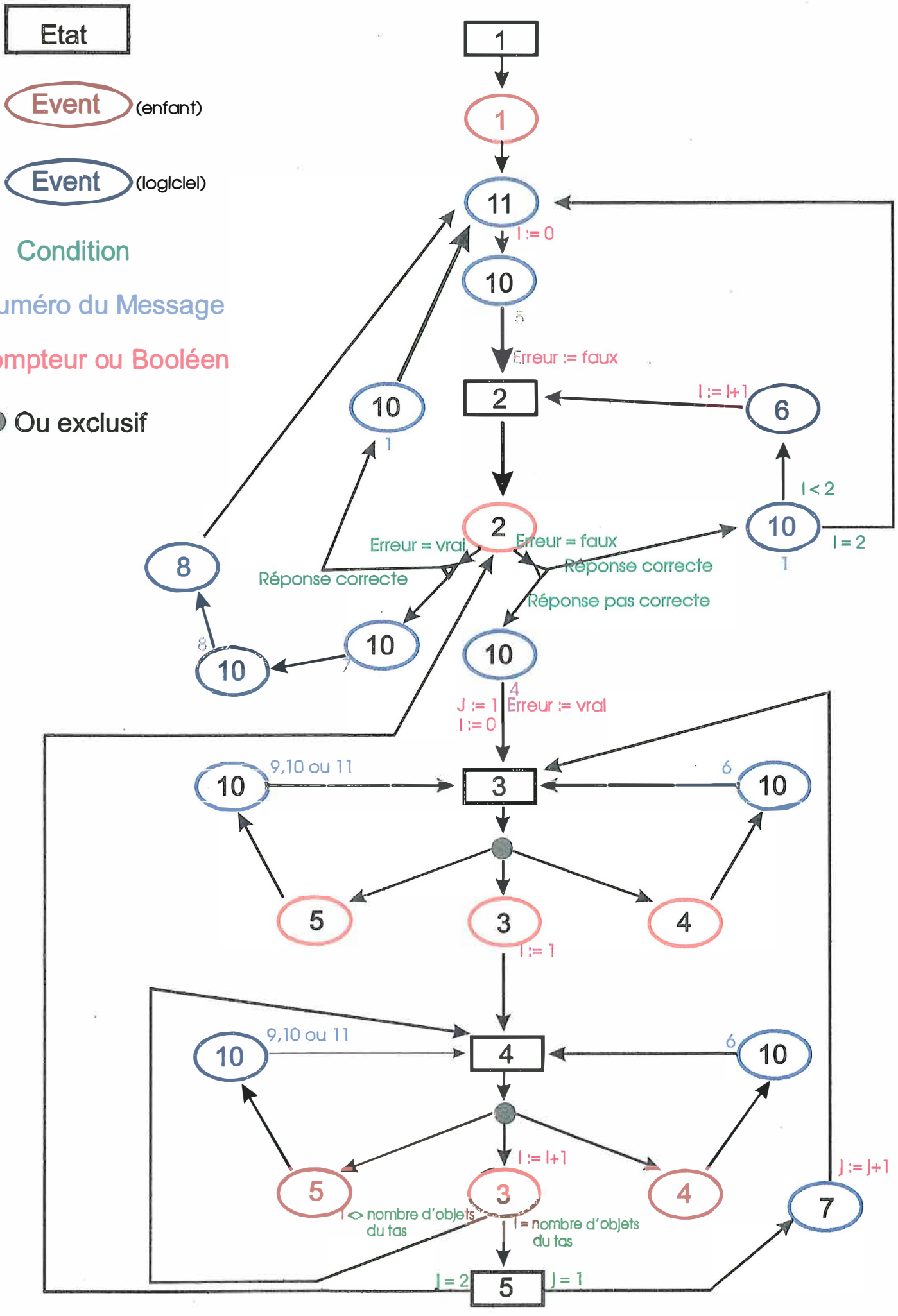
Il existe deux types de messages. D'une part, les messages qui s'effacent suite à un événement (ex : clique d'un bouton) et d'autre part, les messages qui s'effacent automatiquement après un certain délai. Pour ces derniers, l'utilisateur peut cliquer sur le bouton droit de la souris pour mettre fin prématurément à l'affichage.

Event 10. : Afficher deux nouveaux tas.

3) Messages possibles.

Numéro du message	Message
1	"BRAVO !!! Tu as donné la bonne réponse."
2	"Compte les objets du tas A en les cliquant. Quand tu as fini, tu cliques le bouton 'Fini de compter'."
3	"Compte les objets du tas B en les cliquant. Quand tu as fini, tu cliques le bouton 'Fini de compter'."
4	"ERREUR !!! Tu t'es trompé."
5	"Appuie sur le bouton qui convient."
6	"Tu n'as pas fini de compter les objets du tas."
7	"ERREUR !!! Tu t'es encore trompé !!!"
8	"REGARDE."
9	"Tu as déjà compté cet objet !"
10	"Tu as fini de compter les objets du tas. Clique le bouton 'Fini de compter'."
11	"Tu n'as pas cliqué un objet. Mets la main sur un objet du tas et appuie sur le bouton gauche de la souris."

4) Déroulement.


Etat
Event (enfant)
Event (logiciel)
Condition
Numéro du Message
Compteur ou Booléen
Ou exclusif
1
1
11
I := 0
10
5
Erreur := faux
2
I := I+1
6
10
1
I < 2
2
Erreur = vrai
Erreur = faux
Réponse correcte
Réponse correcte
Réponse pas correcte
8
10
10
8
10
I = 2
1
4
J := 1
I := 0
Erreur := vrai
10
9,10 ou 11
3
6
10
5
3
4
I := 1
10
9,10 ou 11
4
6
10
5
3
4
I := I+1
J := J+1
7
I <> nombre d'objets du tas
I = nombre d'objets du tas
J = 2
5
J = 1

D)Les différents taux.

Lors du développement de la conservation chez l'enfant, deux éléments extrêmement importants interviennent : la mise en correspondance terme à terme et la manipulation.

La mise en correspondance doit être possible quelle que soit la configuration (linéaire/quelconque) des 2 ensembles et que ces ensembles soient identiques ou non.

D'autre part, l'enfant doit comprendre qu'une manipulation autre que l'ajout et la suppression d'un objet ne modifie pas la quantité d'un ensemble.

L'application fait intervenir ces deux éléments.

En effet, lorsque l'application doit proposer deux tas d'objets à l'enfant, elle choisit diverses caractéristiques pour les tas dont leur configuration et le fait que ces tas soient identiques ou non.

D'autre part, des manipulations sont réalisées sur ces tas. Les manipulations possibles sont les suivantes :

le changement du type d'objets composant le tas, le changement de la taille des objets, le changement de disposition (écarter/rapprocher les objets), le changement de configuration.

Une dernière manipulation a été ajoutée. Il s'agit de la suppression d'un objet. Cette dernière a été ajoutée afin d'éviter que l'enfant ne donne la bonne réponse uniquement parce qu'il s'est rendu compte qu'après une manipulation, la réponse est la même que la précédente. Cette manipulation a donc pour but d'empêcher un automatisme ignorant.

Lorsque l'application doit choisir une caractéristique pour un tas d'objets ou une manipulation, elle le fait de manière aléatoire. Cependant, pour les deux caractéristiques essentielles que sont la configuration et l'équivalence ainsi que pour le choix d'une manipulation, l'application se base sur des données. Ces données sont les taux d'échecs pour ces caractéristiques et pour les manipulations et font en sorte que l'application choisisse surtout les caractéristiques et les manipulations pour lesquelles ces taux d'échecs sont élevés c'est-à-dire pour lesquelles l'enfant éprouve des difficultés.

L'application prend en compte 4 types de taux d'échecs et l'utilisateur peut choisir sur quel type de taux l'application doit se baser au moment de choisir une caractéristique/manipulation.

Les 4 types de taux sont les suivants :

Les taux d'échecs utilisateur, les taux d'échecs globaux, les taux d'échecs pour les X derniers résultats et les taux d'échecs pondérés.

Les taux d'échecs utilisateur.

Il s'agit de taux fixés par l'utilisateur. Par défaut, tous ces taux valent 0.5. c'est-à-dire que pour chaque caractéristique/manipulation, on considère que l'enfant donne une bonne réponse sur deux. La conséquence de l'égalité des taux est que chaque caractéristique/manipulation a la même probabilité d'être choisie et présentée à l'enfant.

Ces taux peuvent être modifiés. Ils peuvent prendre une valeur comprise entre 0 et 1.

Ils permettent notamment à l'utilisateur de forcer l'application à choisir pratiquement toujours la même caractéristique/manipulation. Il suffit de donner au taux correspondant à cette caractéristique/manipulation la valeur 1 et de donner aux autres caractéristiques/manipulations la valeur 0.

Les taux d'échecs globaux.

Le taux d'échecs global pour la caractéristique/manipulation α est le rapport entre le nombre de mauvaises réponses données par l'enfant et le nombre total de réponses données.

Les taux d'échecs pour les X derniers résultats.

Le taux d'échecs pour les X derniers résultats pour la caractéristique/manipulation α est le rapport entre le nombre de mauvaises réponses données par l'enfant lors des X dernières réponses pour la caractéristique/manipulation α et le nombre X.

X est un paramètre de l'application.

Les taux d'échecs pondérés.

Le taux d'échecs pondéré pour la caractéristique/manipulation α est une pondération entre le taux global et le taux pour les X derniers résultats pour la caractéristique/manipulation α. La pondération s'effectue selon la formule suivante :

$TG \times (1-\alpha) + TD \times \alpha$ où

TG = Taux d'échecs global
TD = Taux d'échecs pour les X dernières réponses.
α = le coefficient de pondération.

α peut être modifié par l'utilisateur.

E) Les menus.

1) Le menu Fichier.

Nouveau

Permet à l'utilisateur d'entrer son nom et son prénom (voir Boîtes de dialogue). Si l'utilisateur a déjà réalisé un certain nombre d'exercices, les résultats sont annulés. Les taux d'échecs globaux, pour les X dernières réponses et pondérés sont désactivés, seuls les taux utilisateur sont actifs et valent 0.5, la valeur par défaut.

Ouvrir

Permet à l'utilisateur de reprendre une session avec des résultats préalablement sauvés.

Sauver

Permet à l'utilisateur de sauver ses différents résultats de sorte qu'il puisse par la suite recommencer une nouvelle session en se basant sur ces résultats.

Quitter

Permet à l'utilisateur de mettre fin à une session de l'application.

2) Le menu Exercice.

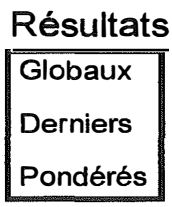
Commencer

Permet à l'utilisateur de lancer un exercice. Cette commande n'a d'effet que si aucun exercice n'est en cours d'exécution.

Terminer

Permet à l'utilisateur de mettre fin à un exercice.

3) Le menu Résultats.



Globaux

Permet à l'utilisateur de prendre connaissance de ses divers taux d'échecs globaux.

Derniers

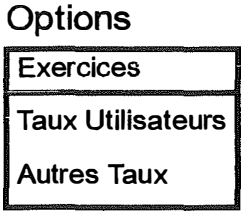
Permet à l'utilisateur de prendre connaissance de ses divers taux d'échecs pour les X dernières réponses.

Pondérés

Permet à l'utilisateur de prendre connaissance de ses divers taux d'échecs pondérés.

Ces trois items ne sont actifs que lorsqu'il y a au moins X résultats pour chaque type d'exercices/manipulations, X étant un paramètre du logiciel qui peut être modifié en choisissant Options/Autres Taux.

4) Le menu Options.



Exercices

Permet de modifier un certain nombre de paramètres du logiciel. Consultez 'Boîte 'Options pour les exercices' ' de la section 'Les boîtes de dialogue' pour connaître les paramètres modifiables en choisissant cet item.

Taux Utilisateur

Permet de modifier un certain nombre de paramètres du logiciel. Consultez 'Boîte 'Taux utilisateur' ' de la section 'Les boîtes de dialogue' pour connaître les paramètres modifiables en choisissant cet item.

Autres Taux

Permet de modifier un certain nombre de paramètres du logiciel. Consultez 'Boîte 'Autres taux' ' de la section 'Les boîtes de dialogue' pour connaître les paramètres modifiables en choisissant cet item.

F Les boîtes de dialogue.

Vous trouverez l'aspect de ces boîtes à la fin de ce document (Annexe).

1) Boîte 'Bienvenue'.

Cette boîte est utilisée lorsque l'utilisateur choisit Fichier/Nouveau. Elle permet à l'utilisateur d'entrer son nom et son prénom.

2) Boîte 'Afficher Taux'.

Indique le nom de l'utilisateur, son prénom ainsi que le taux d'échecs par type d'exercices/manipulations[5]. Chaque contrôle de saisie ne permet que l'affichage de données et non leur modification. La boîte ne comporte pas de titre car elle est utilisée aussi bien pour afficher les taux globaux que les taux pour les X dernières réponses et les taux pondérés.

---

[5] Les types d'exercices possibles sont les suivants :
. Les 2 tas sont identiques.
. Les 2 tas ne sont pas identiques.
. Les 2 tas ont une configuration linéaire (horizontale).
. Les 2 tas ont une configuration quelconque.

Les (types de) manipulations possibles sont les suivantes :
. Changer le type d'objets.
. Changer la taille des objets.
. Rapprocher les objets.
. Ecarter les objets.
. Passer d'une configuration linéaire à une configuration quelconque ou inversément.
. Supprimer un objet.

3) Boîte 'Taux Utilisateur'.

Cette boîte comporte les mêmes contrôles que la précédente si ce n'est que les boîtes de saisies relatives aux types d'exercices/manipulations permettent l'affichage et la modification des différents taux.

4) Boîte 'Options pour les exercices'.

Contrôles de saisie.

Le contrôle de saisie situé à coté de 'Nombre d'objets minimum par tas' permet à l'utilisateur d'entrer un entier positif compris entre 1 et 5 qui sera le nombre minimum d'objets pour les tas.

Le contrôle de saisie situé à coté de 'Nombre d'objets maximum par tas' permet à l'utilisateur d'entrer un entier positif compris entre 1 et 15 qui sera le nombre maximum d'objets pour les tas.

Remarques :

Il faut évidemment que le nombre maximum d'objets soit supérieur ou égal au nombre minimum d'objets.

D'autre part, le nombre d'objets maximum doit être tel qu'il soit possible d'afficher un tas d'objets linéaire comprenant ce nombre d'objets dans la fenêtre 'Fenêtre tas'. Le nombre d'objets maximum dépend de la résolution de votre écran. Un nombre d'objets maximum trop grand risque de provoquer une erreur d'exécution du type 201 (débordement d'intervalle).

Cases à cocher.

La case à cocher 'Historique actif' active ou désactive l'historique. L'historique est un mécanisme qui garde en mémoire 4 entiers qui sont les valeurs représentant les cardinaux des 4 derniers tas proposés à l'enfant (avant toute manipulation) de sorte qu'un même nombre d'objets n'apparaisse trop fréquemment. Si vous activez le mécanisme, vous devez vous assurer que le nombre d'objets maximum moins le nombre d'objets minimum est supérieur ou égal à 4. Si ce n'est pas le cas, l'application risque de bloquer le système.

La case à cocher 'Manipulations visibles' active ou désactive l'aspect visuel d'une manipulation. Dans le cas d'un changement de configuration d'un tas, si vous cochez la case, au lieu de voir directement le tas sous sa nouvelle configuration, vous voyez les objets du tas se déplacer de leur ancienne position vers leur nouvelle.

La case à cocher 'Affichage message réussite' active ou désactive l'affichage des messages en cas de réussite de la part de l'utilisateur.

Boutons radio.

Les 4 boutons radio permettent à l'utilisateur d'indiquer sur quels taux d'échecs l'application doit se baser pour choisir une caractéristique/manipulation.

5) Boîte 'Autres Taux'.

Cette boîte comporte deux contrôles de saisie.

Le premier permet à l'utilisateur de fixer le nombre de dernières réponses que l'on garde en mémoire pour chaque caractéristique/manipulation afin de calculer les taux pour les X (la valeur du paramètre) dernières réponses.

Le second permet à l'utilisateur d'entrer un coefficient de pondération (α)[6].

[6] Le ième taux pondéré est égal au ième taux global multiplié par (1-α) + le ième taux pour les X derniers résultats multiplié par α.

19. Comment ajouter de nouveaux types d'éléments utilisables par le didacticiel ?

Pour ajouter un type d'éléments, par exemple 'EBalle' représentant une balle multicouleur, il convient de procéder de la sorte :

Etape 1. Créer trois nouvelles icônes.

Vous devez créer trois nouvelles icônes (appelées icônes-éléments) représentant trois balles de tailles différentes (maximum, moyenne, minimum). Pour connaître ce qu'on entend par taille maximum, moyenne et minimum, référez-vous, par exemple, aux trois icônes-éléments TomateMax, TomateMoy et TomateMin représentant une tomate dans ces trois tailles. Il faut que votre balle de taille maximum (moyenne,minimum) occupe approximativement la même surface que le motif représentant une tomate dans sa taille maximale (moyenne,minimale) afin que l'enfant ne puisse percevoir une différence de taille entre les deux.

Nous vous renvoyons au 'Guide de l'utilisateur' de Resource Workshop pour prendre connaissance de la manière de créer une icône.

Remarque : Le type d'éléments 'EBalle' étant utilisé dans l'application, vous n'avez pas besoin de créer les trois icônes. Deux autres ensembles de trois icônes-éléments représentant respectivement des carrés et des rectangles et non utilisés dans l'application sont également définis. Vous pouvez, par exemple, essayer de les utiliser dans l'application.

Etape 2. Renommer les icônes et leur affecter des identificateurs.

Une fois que vous avez créé vos trois icônes-éléments, renommez-les de façon à vous souvenir laquelle des trois icônes-éléments représente l'élément dans sa taille maximale (moyenne,minimale).

Ex : BalleMax,BalleMoy,BalleMin

Lorsque vous renommez une icône, Resource Workshop vous demande si vous désirez lui affecter un identificateur. Choisissez 'Oui' ('Yes').

Entrez alors une valeur.

Les identificateurs des 3 icônes-éléments doivent respecter les 3 conditions suivantes:

- Etre consécutifs.
- Le plus petit est associé à l'icône représentant l'élément sous sa taille maximum et le plus grand, l'élément sous sa taille minimum.
- Le plus petit doit être un multiple de 3.

Dans notre cas, les identificateurs sont les suivants :

Nom de l'identificateur	Valeur
BalleMax	114
BalleMoy	115
BalleMin	116

Etape 3. Modifier le module 'Globaux'.

Six choses doivent être modifiées et d'autres doivent être ajoutées.

1) Les modifications.

a) Le type énuméré TETypesElements.

Situation avant modification :

TETypesElements = (ETomate,ECitron,ERien,EMultiType);

Situation après modification :

TETypesElements = (ETomate,ECitron,EBalle,ERien,EMultiType);

La modification apportée consiste à ajouter une nouvelle constante du type TETypesElements (dans notre cas, nous l'avons appelée EBalle).

b) La constante CNbrElementsDefinis.

Il s'agit d'incrémenter la constante CNbrElementsDefinis d'une unité pour chaque nouvelle constante du type TETypeElements.

Dans notre cas, la constante est passée de la valeur 2 à la valeur 3.

c) La constante tableau CTabIndicesElements

Pour rappel, ce tableau reprend le plus petit identificateur pour chaque type d'éléments ainsi que les constantes 0,-1 pour respectivement les constantes du type TETypesElements ERien et EMultiType.

Pour chaque nouvelle constante du type TETypesElements, il convient d'ajouter le plus petit des identificateurs associés à un type d'éléments. L'ajout de cette identificateur ne peut se faire n'importe comment. Si la nouvelle constante du type énuméré TETypesElements occupe la $i^{ème}$ position dans l'énumération, l'identificateur ajouté au tableau CTabIndicesElements doit également occuper la $i^{ème}$ position dans ce même tableau.

Nous avons donc les situations suivantes :

Avant modification :

CTabIndicesElements:TTableauIndicesElements=(TomateMax,CitronMax,0,-1)

Après modification :

CTabIndicesElements:TTableauIndicesElements=(TomateMax,CitronMax,BalleMax,0,-1)

d) La constante CLongTabIndicesElements

Pour chaque nouvel identificateur ajouté au tableau CTabIndicesElements, il faut incrémenter la constante d'une unité.

Dans notre cas, la constante passe de 4 à 5.

e) La fonction FRetournerType

La fonction doit être modifiée de façon à ce que lorsqu'on lui donne, comme valeur du paramètre Index, le rang d'une constante du type TETypesElements, celle-ci renvoie, via le paramètre UnTypeElement, la constante appropriée du type TETypesElements.

A chaque fois qu'une nouvelle constante est ajoutée au type énuméré TETypesElements à la $i^{ème}$ position dans l'énumération, la ligne suivante doit être ajoutée à la $i^{ème}$ position dans l'instruction **case :**

```
i : TypeElements := EBalle;
```

Les autres lignes de l'instruction **case** suivant cette ligne doivent avoir leur constante de cas augmentée d'une unité.

Dans notre cas, nous avons les situations suivantes :

Situation avant modification :

```
function FRetournerTypeElements;

begin
FRetournerTypeElements := TRUE;
case Index of
   0 : TypeElements := ETomate;
   1 : TypeElements := ECitron;
   2 : TypeElements := ERien;
   3 : TypeElements := EMultiType;
   else FRetournerTypeElements := FALSE;
   end;
end;
```

Situation après modification :

```
function FRetournerTypeElements;

begin
FRetournerTypeElements := TRUE;
case Index of
   0 : TypeElements := ETomate;
   1 : TypeElements := ECitron;
   2 : TypeElements := EBalle;
   3 : TypeElements := ERien;
   4 : TypeElements := EMultiType;
   else FRetournerTypeElements := FALSE;
   end;
end;
```

f) La fonction FCreerRegion.

Pour rappel, cette fonction est capable de créer une région au sens Windows 'couvrante' pour chaque icône définie. Il faut donc modifier la fonction de façon à ce qu'elle soit capable de créer une région pour chaque nouvelle icône.

Cela se fait très facilement. Il suffit d'ajouter trois lignes dans l'instruction case, chacune ayant la syntaxe suivante :

```
Identificateur de l'icône :  FCreerRegion :=createpolygonrgn(CTabRegionXXX,CLongTabRegionXXX, ALTERNATE);
```

où CTabRegionXXX et CLongTabRegionXXX sont de nouvelles constantes ajoutées au module Divers (voir section suivante).

2) Les ajouts.

Pour chaque nouvelle constante du type TETypesElements, il convient d'ajouter au module Divers 6 nouvelles constantes.

Ces constantes permettent de créer des régions au sens Windows qui couvrent totalement les motifs des trois nouvelles icônes.

Trois constantes sont du type TTabCoord qui représente un tableau de coordonnées, les trois autres sont des constantes entières qui indiquent les longueurs effectives des trois tableaux.

On a donc une constante du type TTabCoord pour chaque nouvelle icône-élément. Une donnée de ce type de tableau est une paire d'entiers représentant les coordonnées d'un point à l'intérieur d'une icône 32x32. Ces paires d'entiers (que nous appellerons points par la suite) doivent être telles que si on relie la $i^{ème}$ à la $i+1^{ème}$ et la dernière à la première, on obtient une figure fermée englobant totalement le motif de l'icône traitée. De plus, le contour de la figure doit être le plus proche possible du contour du motif de l'icône-élément.

Comment construire ces tableaux ?

Le procédé est simple. Lancez Resource Workshop, choisissez le projet approprié puis choisissez l'icône pour laquelle vous souhaitez créer une région (par exemple BalleMax).

Etape 1. Choisir un point extérieur au motif de l'icône. Vous pouvez apercevoir les coordonnées de ce point au bas de votre écran. Il s'agit de la première donnée du tableau du type TTabCoord, que nous avons appelé CTabRegionBalleMax.

Etape 2. Choisir un autre point toujours extérieur au motif de l'icône et telle que la droite joignant ce point au précédant ne soit pas en contact avec le motif de l'icône tout en étant le plus proche possible du contour du motif. Les coordonnées de ce point constituent la $i^{ème}$ (i=2..CLongTabRegionBalleMax qui indique la longueur effective du tableau) donnée du tableau CTabRegionBalleMax.

Etape 3. Réitérer l'étape 2. jusqu'à ce que le nouveau point choisi soit tel que si on trace une droite de ce point au premier, on obtient une figure fermée englobant la totalité du motif de l'icône.

Si le nombre de points nécessaires à la création de la figure est inférieur à 16, vous complétez le tableau avec la valeur (x:0;y:0).

Vous avez alors construit la constante tableau CTabRegionBalleMax qui contient un ensemble de points représentant une figure englobant la totalité du motif de l'icône-élément BalleMax.

Vous devez alors créer la constante CLongTabregionBalleMax et lui affecter une valeur entière indiquant le nombre de points nécessaires pour créer la figure englobante.

Vous faites de même pour les icônes BalleMoy et BalleMin.

Une fois ce travail réalisé, les nouvelles icônes-éléments seront utilisées par l'application.

Partie 5 : BIBLIOGRAPHIE.

Références pédagogiques.

Piaget J. et Szeminska A., 1941, *' La genèse du nombre chez l'enfant'*, Neufchâtel, Delachaux et Niestlé, 1967.

Piaget J., *'Six études de psychologie'*, Paris, Editions Denoël Gonthier, 1964.

Deliège M. et Botson C., Tableaux synoptiques des expériences menées par Piaget publiés par la Direction générale de l'organisation des études dans la collection "Pédagogie et Recherche", 1974.

Brissiaud R., *'Comment les enfants apprennent à calculer'*, Paris, Editions Retz,1989.

L'école fondamentale, Dossier 13, Namur, Editions Scolaires ERASME, Janvier/Fevrier 87.

L'école fondamentale, Dossier 14, Namur, Editions Scolaires ERASME, Mars/Avril 87.

Roegiers X., *'Guide systhématique de base'* , Bruxelles, Editions de Boeck, 1985.

Andrianne L. et Dupagne J., *'Matthmo'*, Livre 1a, Liège, Editions H. Dessain,1971.

Bernard C., *'Pedagogie expérimentale. La genèse du nombre d'après Piaget. Applications pratiques àl'école maternelle.'*, Travail de fin d'étude, 1983.

Conseil de la Communauté Française, Circulaire 12, Bruxelles, Réf : AC/CS/25.001, Août 1995.

Références informatiques.

Borland International, Borland Pascal Objets (version 7.0), '*Guide du programmeur*'.

Borland International, Borland Pascal Objets (version 7.0), '*Guide de référence*'.

Borland International, ObjectWindows, '*Guide du programmeur*'.

Borland International, Resource Workshop, '*Guide de l'utilisateur*'.

Borland International, Turbo Debugger, '*Guide de l'utilisateur*'.

TABLE DES MATIERES.