MASTER EN SCIENCES INFORMATIQUES

Projet d'aide à la compréhension de l'arithmétique fondamentale : la multiplication et la division

Walthéry, Valérie

Award date:
1995

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de le Paix
Institut d'Informatique

Rue Grandgagnage, 21, B- 5000 Namur (Belgium)
Année académique 1994-1995.

Projet d'aide à la compréhension de l'arithmétique fondamentale : la multiplication et la division

Mémoire de fin d'études
présenté par Valérie Walthéry
en vue de l'obtention du grade de
licenciée et maître en informatique.

Promoteur : Monsieur Claude Cherton

Mes remerciements vont tout d'abord à mon promoteur, Monsieur Claude Cherton, qui m'a assisté tout au long de l'élaboration de ce mémoire et qui m'a donné de nombreux conseils.

Je tiens ensuite à remercier Monsieur Christian Esch pour l'aide précieuse qu'il m'a apportée lorsque j'en avais besoin.

Enfin, mes pensées vont à mes parents et mes amis pour leur soutien et leurs encouragements.

Résumé

Ce mémoire poursuit les recherches sur l'aide à la compréhension de l'arithmétique fondamentale et envisage plus particulièrement les opérations de multiplication et de division. L'objectif à long terme de ce mémoire est l'élaboration d'un générateur de didacticiels qui premettrait aux enseignants de construire leurs propres didacticiels.

Pour arriver à un tel générateur, nous proposerons une bibliothèque d'outils que le générateur pourra utiliser. Nous exposerons également un gestionnaire d'événements qui est un premier pas vers l'élaboration du générateur de didacticiels.

Les différents chapitres reprennent successivement une analyse sous l'aspect pédagogique des opérations de multiplication et de division, l'analyse logicielle des objets à construire pour permettre l'apprentissage de ces opérations et un exemple concret d'utilisation des objets créés dans un programme.

Abstract

This thesis carries out researches into the help of comprehension of the basic arithmetic and deals specially with the multiplication and division operations. The long-term purpose of this thesis is the elaboration of a generator of educational software programs which would enable the teachers to build their own educational software.

We'll give a library of programmed objects which could be used by the generator. The manager of events proposed in this thesis is a first step to the elaboration of the generator.

The different chapters deal successively with the pedagogical aspect of the multiplication and division operations, the software analysis of the objects to be build to enable the learning of these operations and a concrete example of the objects built inside a program.

TABLE DES MATIERES

1

INTRODUCTION

1.1. Pourquoi un didacticiel ?

L'ordinateur prend une place de plus en plus importante dans notre vie quotidienne. Il est en effet présent dans la plupart des lieux que nous fréquentons et, les entreprises, les administrations et même les petits commerçants ne peuvent désormais plus s'en passer.

L'ordinateur est également présent dans les écoles, mais, celles-ci ne disposant que de budgets restreints, l'achat d'ordinateurs ne constitue pas une priorité. L'utilisation de l'ordinateur comme aide à l'apprentissage est dès lors très limitée, l'écolier préférant de plus les jeux et les applications commerciales aux didacticiels.

Pourtant, les didacticiels qui sont des logiciels spécialement conçus pour l'enseignement assisté par ordinateur, constituent une manière différente et attrayante pour les enfants d'apprendre. L'ordinateur étant nouveau et inconnu pour la plupart d'entre eux, la curiosité

qu'ils portent à cette machine peut stimuler leur envie d'apprendre. De plus, le côté ludique des didacticiels rend l'apprentissage moins fastidieux, les enfants apprenant en s'amusant. Apprendre devient donc un jeu. C'est un des avantages de l'enseignement assisté par ordinateur : s'amuser en travaillant. L'affichage rapide et l'analyse des réponses de l'utilisateur, la capacité de fournir des exercices différents (en changeant simplement les valeurs numériques d'un calcul), ...sont aussi des avantages de l'enseignement assisté par ordinateur.

S'il est bien conçu, un didacticiel peut apporter beaucoup aux élèves mais aussi aux enseignants. Certains enfants, ayant des difficultés pour apprendre, peuvent trouver dans un didacticiel le moyen de combler leurs lacunes. De même, certains enseignants, ayant tout essayé pour aider les élèves plus faibles, trouveront peut-être la solution à leurs problèmes en les laissant « jouer » avec l'ordinateur.

Néanmoins, l'ordinateur ne peut remplacer l'intuition et la présence d'un instituteur, ni comprendre les rapports humains. Si l'élève a des difficultés à apprendre à cause de problèmes familiaux ou psychologiques, l'instituteur pourra le voir ; l'ordinateur pas ! De plus, l'ordinateur peut avoir des difficultés à diagnostiquer une erreur, alors que l'instituteur, grâce à son expérience et à son intelligence, aura plus de chances d'y arriver. Les enseignants devraient donc travailler en parallèle avec les ordinateurs car ils peuvent se compléter. L'idéal serait même de laisser les enseignants façonner leurs didacticiels pour qu'ils puissent y apporter leur expérience.

Un bon didacticiel doit apprendre certaines notions à l'élève, mais il doit aussi lui permettre d'en découvrir par lui-même. Il doit être attrayant, amusant, pour que l'enfant ait envie de l'utiliser et d'apprendre. Il doit fournir des explications appropriées et utiles pour l'enfant. Il doit, en plus, évoluer avec l'élève.

En effet, les besoins en explications évoluent avec le niveau et l'expérience de l'élève. De plus, un élève de bon niveau n'a pas besoin des mêmes instructions et explications qu'un élève de niveau faible. Si l'on donne trop d'explications inutiles à un élève, il se lassera vite du didacticiel. De même, si l'on donne trop peu d'explications à un élève qui en a besoin, celui-ci sera vite perdu et abandonnera le logiciel. Un bon didacticiel doit donc être flexible en s'adaptant aux évolutions de ses utilisateurs.

1.2. Où se situe ce travail ?

Ce travail est la continuation des recherches effectuées par Marc Philippe (voir [biblio 4]). Ses recherches portaient sur l'aide à la compréhension de l'arithmétique fondamentale et plus particulièrement sur les opérations d'addition et de soustraction. Je prolonge donc ses recherches car je tente d'apporter une aide aux élèves confrontés aux opérations de multiplication et de division.

En réunissant ces deux travaux, on obtiendra une aide à l'apprentissage des quatre opérations de base. Ces deux travaux pourront être complétés par des recherches en amont des quatre opérations (sur la signification du nombre lui-même) et par des recherches en aval (sur les fractions, les nombres réels, les puissances, ...). Il existe donc encore beaucoup de domaines à explorer.

Ce mémoire, ainsi que celui de Marc Philippe, est basé sur une idée de boîte à outils. Il consiste donc à proposer une bibliothèque d'outils permettant d'apprendre la multiplication et la division au travers d'approches différentes. Mais cette boîte à outils ne sera utile que pour des programmeurs. La plupart des enseignants n'étant pas spécialistes dans ce domaine, cette boîte à outils ne leur sert à rien.

C'est pourquoi, j'ai introduit l'idée de générateur de didacticiels. Un générateur de didacticiels est une boîte à outils conçue pour les enseignants et non pour les programmeurs. Le générateur utiliserait les objets de la boîte à outils pour programmeurs. Mais il proposerait en plus une interface adéquate qui permettrait aux enseignants de les utiliser.

On peut comparer un générateur de didacticiels à un logiciel de dessin si l'on se met dans la peau d'un programmeur. Un logiciel de dessin propose des outils qui permettent de tracer des lignes, des cercles, des rectangles,. On peut choisir la couleur et l'épaisseur des traits et placer les dessins n'importe où sur l'écran. Un générateur de didacticiels, allant dans le sens de l'aide à la compréhension de l'arithmétique fondamentale, comprendrait des outils permettant d'apprendre les opérations de base de plusieurs manières. On pourrait, par exemple, choisir la

représentation des nombres que l'on souhaite, choisir les dialogues entre la machine et les élèves et placer tous ces objets n'importe où sur l'écran.

Mais, sans une interface appropriée, les « non programmeurs » ne pourraient pas se servir de ce générateur. Il faudrait donc aller beaucoup plus loin pour arriver à un générateur de didacticiels pour les enseignants.

Le but de ce mémoire consiste à construire des outils qui serviront ultérieurement à l'élaboration du générateur de didacticiels exposé ci-dessus. Pour illustrer le fonctionnement de ces outils, je proposerai un petit logiciel qui les utilise.

2

ANALYSE SOUS L'ASPECT PEDAGOGIQUE DU DIDACTICIEL

2.1. Les éléments à reprendre de l'addition et de la soustraction.

L'analyse pédagogique de la multiplication et de la division constitue une étape très importante car elle va influencer et guider la conception de la boîte à outils. Il faut se placer dans la peau d'un élève désirant apprendre à multiplier et à diviser, détecter les problèmes auxquels il peut être confronté et tenter de les résoudre. Une partie de cette analyse a déjà été réalisée dans le travail sur l'addition et la soustraction [Biblio 4], nous pouvons donc nous en inspirer.

2.1.1. Conception orientée objet.

L'idée de conception orientée objet et de boîte à outils est essentielle. Cette idée permet de réutiliser les objets construits pour résoudre d'autres problèmes et finalement, construire un générateur de didacticiel qui se servira de ces outils. Les objets, qui seront proposés, permettront aux élèves de s'exercer de manière intelligente. L'ordinateur leur montrera leurs erreurs et leur indiquera la démarche adéquate pour résoudre le problème en question. De plus, les objets seront construits de manière telle que les élèves puissent aborder la multiplication et la division de différentes manières : ils pourront résoudre le calcul écrit d'une multiplication ou d'une division ainsi qu'il est enseigné dans les écoles ou jouer avec la représentation imagée de ces opérations comme cela était possible avec l'addition (voir [Biblio 4]).

2.1.2. Représentation imagée des nombres.

Dans ce mémoire, nous utiliseront la représentation imagée des nombres proposée dans le travail sur l'addition et la soustraction. Cette représentation convient, en effet, très bien pour les quatre opérations de base.

Les représentations suivantes ont été choisies :

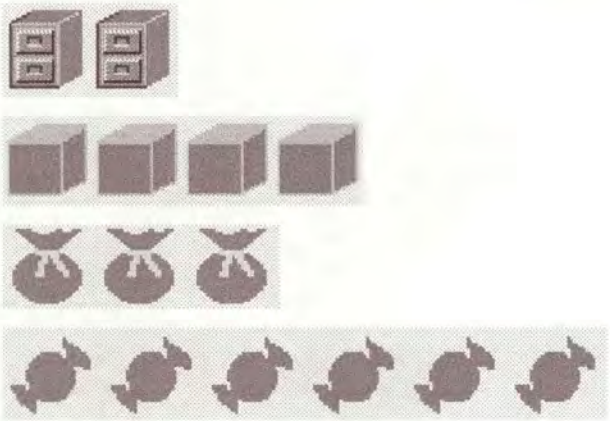
- l'unité est représentée par un « bonbon » :

- la dizaine est représentée par un « sachet » de bonbons :

- la centaine est représentée par une « caisse » :

- et le millier est représenté par une « armoire » :

Par exemple, le nombre « 2436 » est représenté sous forme imagée par 2 « armoires », 4 « caisses », 3 « sachets » et 6 « bonbons » :



Etant donné que nous travaillons avec des nombres naturels, l'unité est indécomposable. Il est donc impossible « d'ouvrir » un bonbon pour en analyser le contenu. Par contre, il est possible d'« ouvrir » un sachet pour obtenir les dix bonbons qui le composent, ou une caisse pour faire apparaître les dix sachets qu'elle contient ou une armoire pour obtenir dix caisses, si nous travaillons en base dix. Les bonbons, sachets, caisses et armoires sont donc des objets réels.

Le choix de cette même représentation permet de ne pas embrouiller les élèves et de garder une certaine continuité dans les quatre opérations de base.

2.1.3. Les éléments nouveaux.

Toutefois, le didacticiel que nous proposons n'est pas une copie de celui consacré à l'addition et la soustraction [Biblio 4]. En effet, les objets de base que nous avons définis sont conçus différemment : les bonbons, les sachets, les caisses et les armoires sont maintenant définis comme des objets informatiques, ce qui n'était pas le cas antérieurement. Cette optique permet de faire connaître à chaque objet de base les actions que l'on peut effectuer sur lui. Par exemple, un objet « bonbon » sait qu'il ne peut pas être ouvert et un objet « sachet » sait qu'il peut être ouvert et qu'il contient 10 objets « bonbon ». L'action d'ouverture d'un objet de base sera donc reprise dans sa définition.

Les objets de base étant différents, les objets d'addition ont été entièrement redéfinis. En effet, la multiplication utilise l'addition et l'addition utilise les objets de base. Nous avons donc reconstruit les objets permettants d'effectuer l'addition puisqu'ils se servent des objets de base. Les objets représentants cette opération ainsi que celle de multiplication et les objets de base seront détaillés plus loin.

Le gestionnaire d'événements, qui sera exposé au point 2.3.5 est également neuf. Il permettra de répondre aux événements provoqués par l'interaction entre la machine et l'élève. Les enseignants définiront eux-mêmes les messages-réponses que le gestionnaire enverra éventuellement lorsqu'un tel événement se produira. Le gestionnaire permettra également de déterminer les conditions d'affichage de ces messages. Il tiendra notamment compte du niveau des connaissances de l'élève, de l'état d'avancement de l'opération que ce dernier effectue, de la fréquence des erreurs qu'il commet,... Les enseignants pourront donc écrire eux-mêmes les messages qu'ils veulent afficher de manière à mieux les adapter à chaque situation. Ils pourront aussi les modifier à souhait grâce à un éditeur de messages.

Tous les éléments nouveaux apportés dans ce travail seront détaillés au point 2.3.

2.2. Analyse pédagogique.

2.2.1. L'apprentissage de la multiplication :

L'analyse du déroulement d'une multiplication écrite montre qu'il y a plusieurs étapes que les élèves doivent franchir pour arriver à multiplier 2 nombres quelconques. Parmi ces étapes, j'en ai relevé 5 qui me paraissent importantes pour un élève qui apprend à multiplier. Ces cinq étapes décrivent une manière d'apprendre la multiplication. Celle-ci n'est évidemment pas unique. Si c'était le cas, un générateur de didacticiel ne serait pas nécessaire. Par cette découpe, j'espère faire ressortir les problèmes que peuvent rencontrer les élèves lors de l'apprentissage de la multiplication.

La première étape consiste à apprendre aux élèves les tables de multiplication, c'est-à-dire, la multiplication de 2 chiffres. Ensuite, avant de passer à la multiplication par 10 et par les multiples de 10, il faut que les élèves aient comprit la multiplication d'un nombre par un chiffre et qu'ils aient remarqué l'utilité du report. Il faut alors généraliser la multiplication à 2 nombres quelconques et bien présenter la décomposition de celle-ci comme une addition de deux multiplications d'un nombre par un chiffre. La multiplication nécessite donc une bonne maîtrise de l'addition.

Je donnerai, pour chaque étape présentée ci-dessus, son but, le type d'exercice qui pourrait être proposé aux élèves, un exemple concret avec des valeurs numériques et le déroulement possible de cet exemple.

J'utiliserai une représentation des nombres plus simple pour exposer l'analyse pédagogique de la multiplication et de la division :

- le bonbon sera schématisé par 
- le sachet sera schématisé par 
- la caisse sera schématisée par 
- l'armoire sera schématisée par 

A. 1° étape :

♦ **But :**

- Apprendre et réviser les tables de multiplication en montrant sous forme imagée le résultat de la multiplication de 2 chiffres.
- Obliger le regroupement pour montrer l'utilité du report.
- Montrer le résultat de la multiplication par zéro.
- Montrer que le résultat da la multiplication est le même si on intervertit les deux chiffres à multiplier.

♦ **Type d'exercice :**

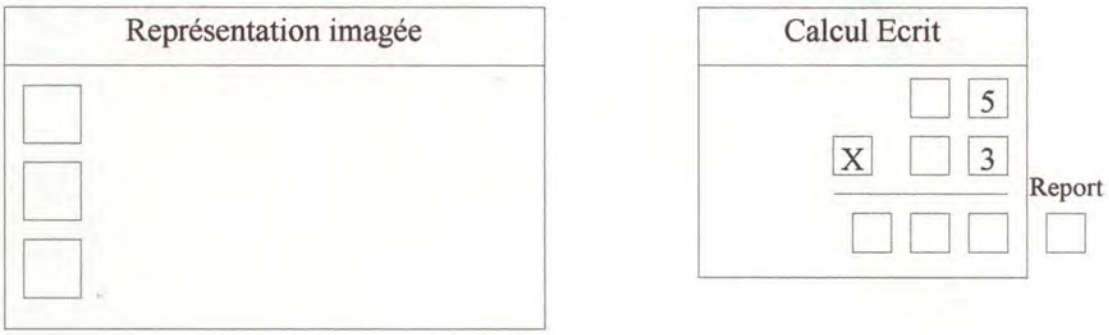
- J'ai X bonbons dans une boîte. Combien ai-je de bonbons si je prends Y boîtes identiques ?
- $X < 10$, $Y < 10$.

♦ **Exemple :**

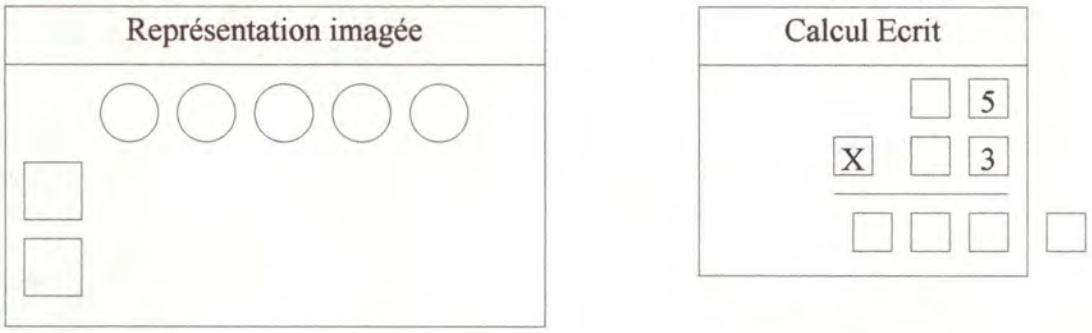
- J'ai 5 bonbons dans une boîte. Combien ai-je de bonbons si je prends 3 boîtes identiques ?

♦ **Déroulement :**

- L'élève sélectionne les deux chiffres à multiplier dans le calcul écrit (ici, il ne peut pas se tromper car il n'y en a que deux).
- La représentation imagée de cette multiplication apparaît (l'élève voit donc apparaître un certain nombre de boîtes).

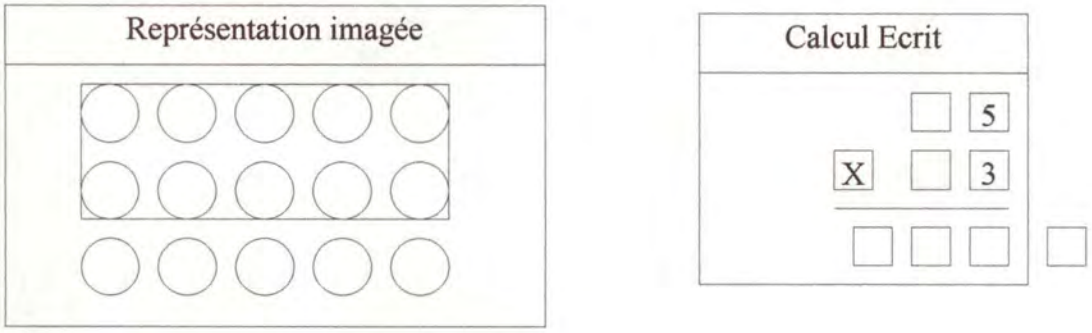


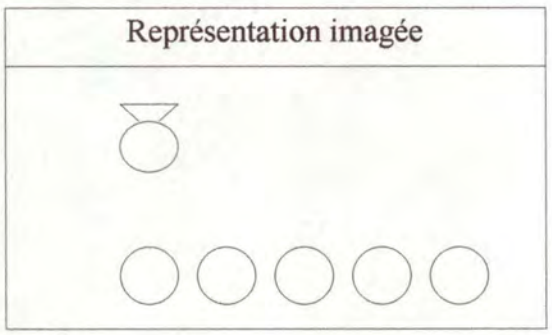
- L'élève peut ensuite ouvrir les boîtes pour faire apparaître les bonbons qu'elles contiennent.

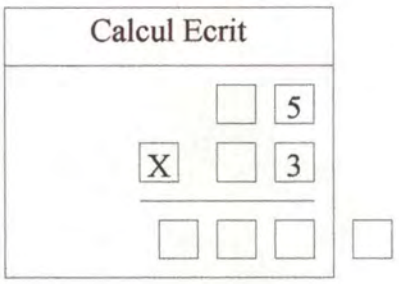


- Quand toutes les boîtes sont ouvertes, il doit effectuer des regroupements de 10 bonbons. Ceux-ci sont remplacés par un sachet. Il fait de même jusqu'à ce que le nombre de bonbons restants soit inférieur à 10. Ce principe de regroupement a été exposé dans le travail sur l'addition et la soustraction [Biblio 4].

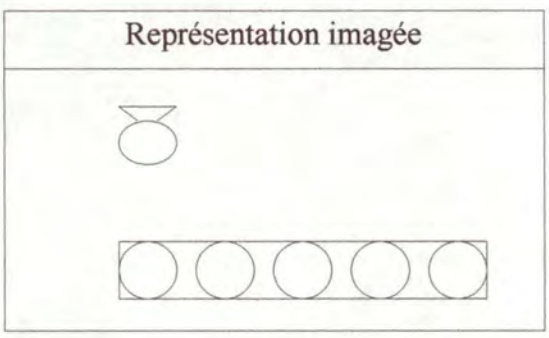
- Le regroupement de 10 éléments de même rang peut poser certains problèmes aux élèves. Il faut donc bien expliciter ce principe.

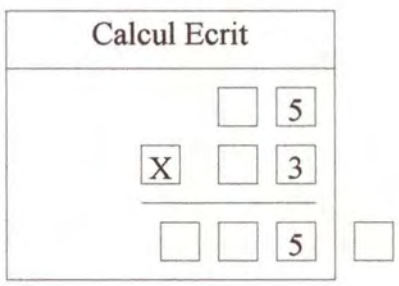




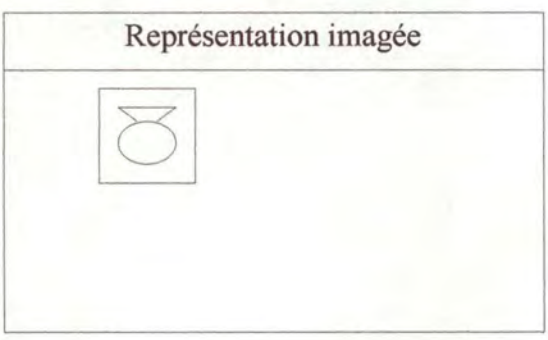


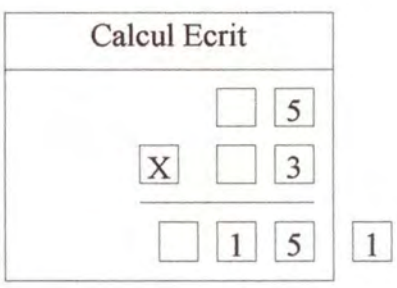
- Il sélectionne ces bonbons restants pour obtenir le résultat au rang des unités dans le calcul écrit.





- Il sélectionne finalement le ou les sachets reports, s'il y en a, pour obtenir le résultat du rang des dizaines dans le calcul écrit. Ce résultat est aussi un report (il faut donc lui faire remarquer).





- Pour montrer l'effet du zéro dans la multiplication, on reprend le même exercice avec :
  - soit 0 boîte qui contient X bonbons ($X > 0$, $Y = 0$) : puisqu'il n'y a aucune boîte à ouvrir, il n'y a aucun bonbon donc le résultat est zéro.
  - soit Y boîtes qui ne contiennent aucun bonbon ($X = 0$, $Y > 0$) : l'élève ouvre les boîtes et ne trouve aucun bonbon donc le résultat est zéro.

- Pour montrer que le résultat d'une multiplication ne change pas si on intervertit les deux nombres à multiplier, il suffit de faire deux exercices consécutifs en interchangeant les valeurs.

B. 2° étape :

♦ **But :**

- Multiplier un nombre par un chiffre.
- Généraliser le système des reports.

♦ **Type d'exercice :**

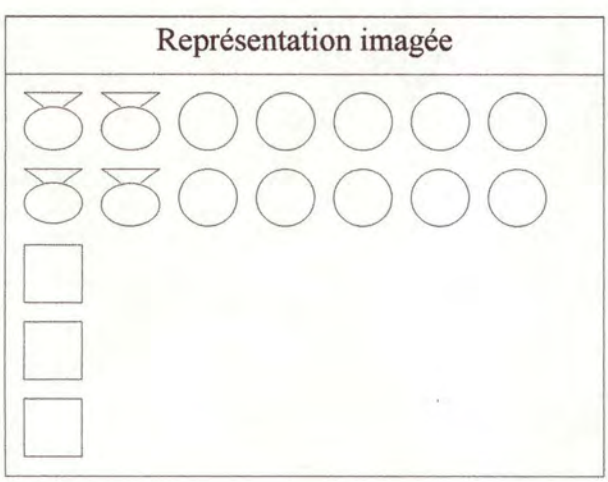
- J'ai X bonbons dans une boîte. Combien ai-je de bonbons si je prends Y boîtes identiques ?
- $X < 100$, $Y < 10$.

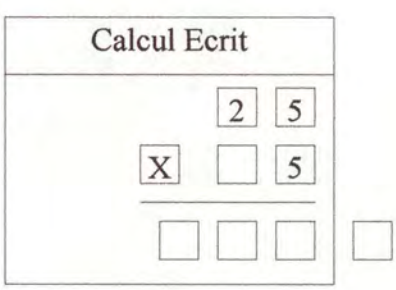
♦ **Exemple :**

- J'ai 25 bonbons dans une boîte. Combien ai-je de bonbons si je prends 5 boîtes identiques ?

♦ **Déroulement :**

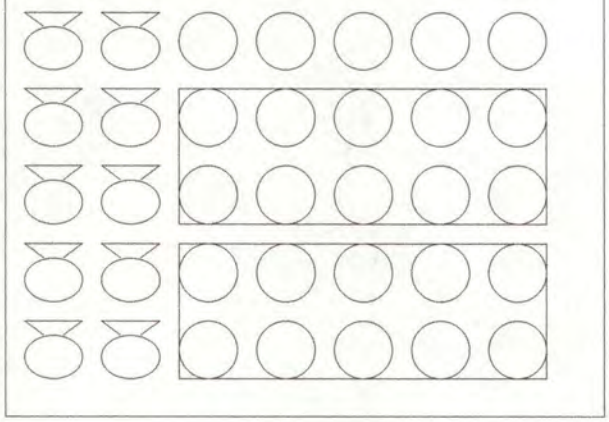
- L'élève sélectionne le chiffre et le nombre à multiplier. Il voit apparaître les 5 boîtes qu'il peut ouvrir pour voir les bonbons et les sachets qu'elles contiennent.



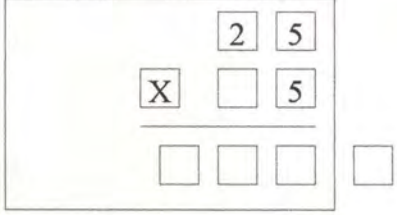


- Quand tous les bonbons et les sachets sont là, il regroupe les bonbons par groupes de 10 pour faire apparaître les sachets reports.

Représentation imagée



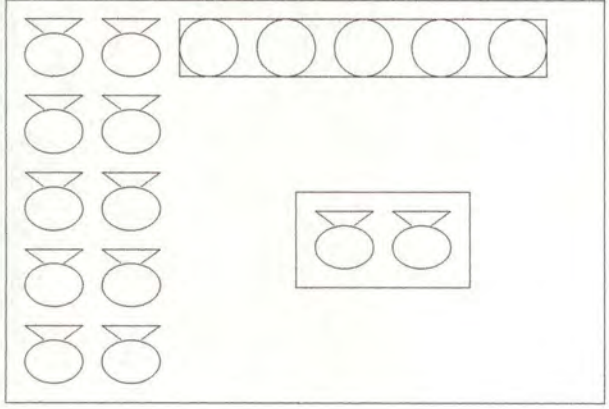
Calcul Ecrit



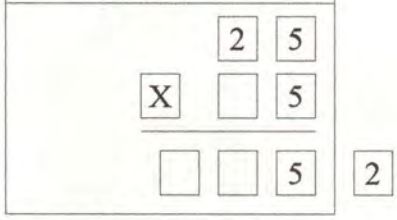
- Il peut ensuite sélectionner les bonbons restants pour faire apparaître le résultat au rang des unités. Il peut aussi inscrire le report dans la case des reports.

- Le report peut aussi poser des problèmes. Par ce type d'exercice nous espérons pouvoir montrer aux élèves le but du report et son utilité.

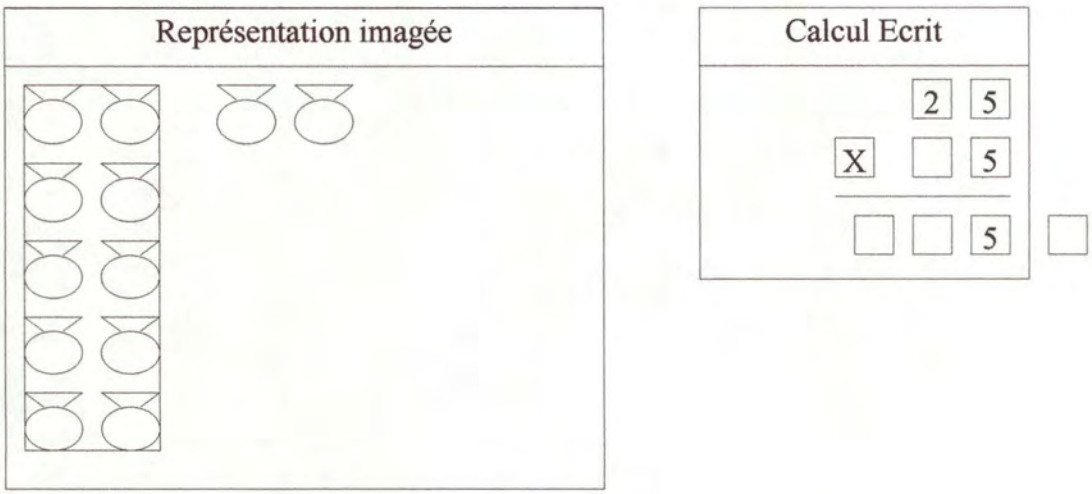
Représentation imagée



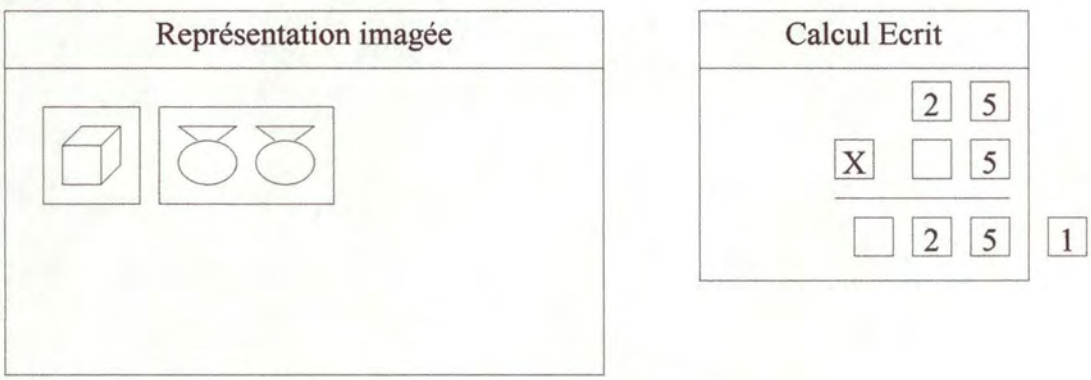
Calcul Ecrit



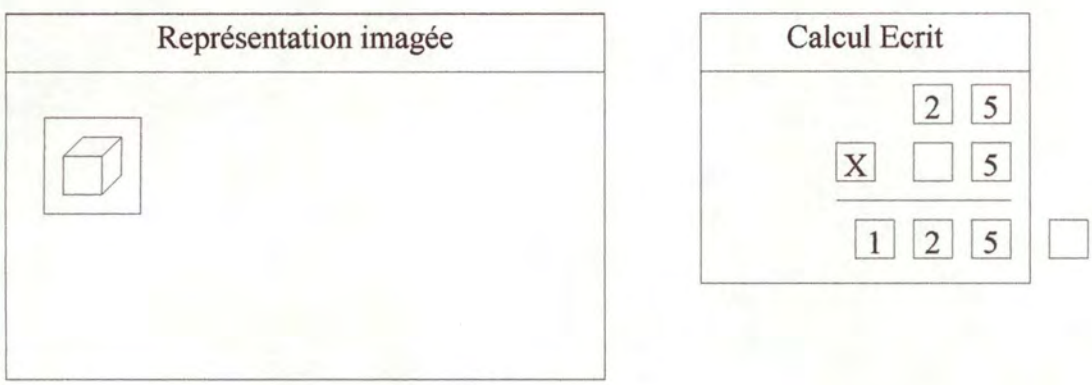
- L'élève doit alors regrouper les sachets par groupes de 10 pour obtenir des caisses, sans oublier d'y ajouter les 2 sachets reports.



- Il doit ensuite inscrire le résultat aux rang des dizaines et le report des centaines dans le calcul écrit.



- Il peut enfin inscrire le résultat au rang des centaines en sélectionnant la caisse report.



C. 3° étape :

♦ **But :**

- Expliquer la multiplication par 10.
- Montrer que multiplier par 10 revient à décaler le multiplicande d'un rang vers la gauche dans le calcul écrit.

♦ **Type d'exercice :**

- J'ai X bonbons dans une boîte. Combien ai-je de bonbons si je prends 10 boîtes identiques ?
- $X < 100$, $Y = 10$.

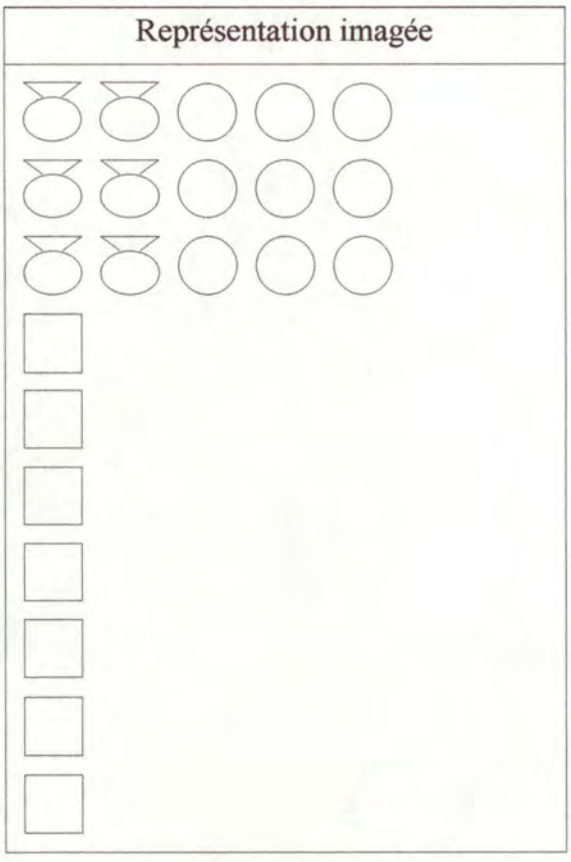
♦ **Exemple :**

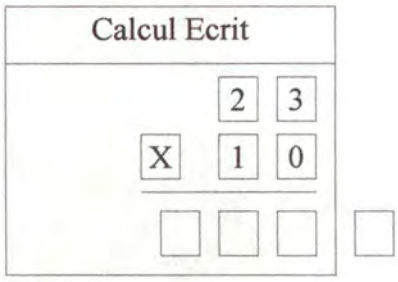
- J'ai 23 bonbons dans une boîte. Combien ai-je de bonbons si je prends 10 boîtes identiques ?

♦ **Déroulement :**

- L'élève sélectionne le multiplicande et le chiffre des dizaines du multiplicateur pour faire apparaître la représentation imagée de la multiplication (c'est-à-dire les dix boîtes qu'il pourra ouvrir).

- Le but de cet exercice est de montrer le résultat d'une multiplication par dix. Pour parvenir ultérieurement à multiplier deux nombres quelconques, les élèves doivent comprendre le principe de la multiplication par dix (décalage du multiplicande d'un rang vers la gauche ou rajout d'un zéro derrière le multiplicande).



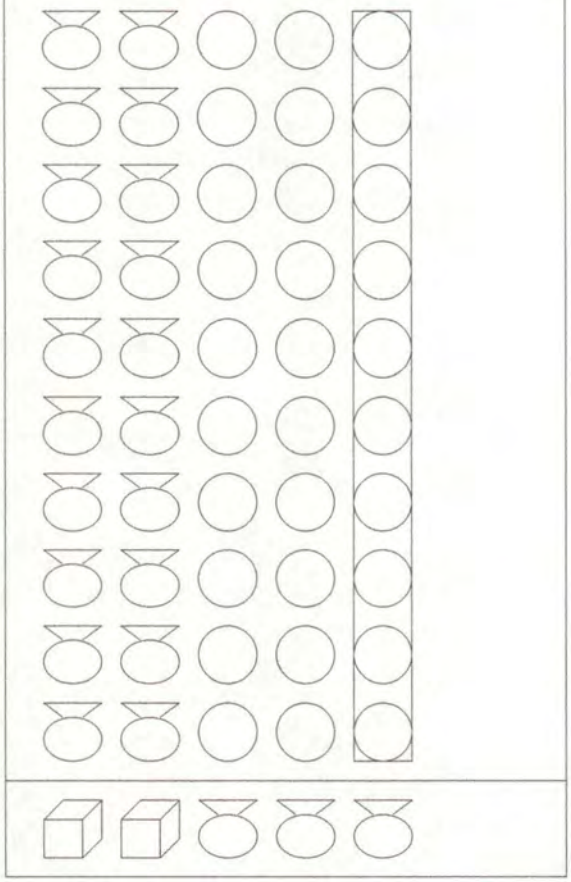


- Quand il a ouvert toutes les boîtes, l'élève regroupe les bonbons verticalement pour obtenir des sachets. Il fait de même avec les sachets pour obtenir des caisses.

- Il peut alors comparer le résultat obtenu avec le nombre initial contenu dans les boîtes. Il remarque qu'il a 2 caisses là où il avait 2 sachets et qu'il a 3 sachets là où il avait 3 bonbons.

- Grâce à cette comparaison, les élèves pourront mieux comprendre le pourquoi du décalage du multiplicande d'un rang vers la gauche.

Représentation imagée

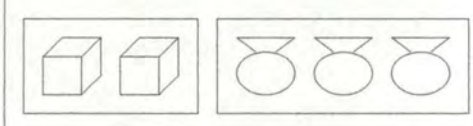


Calcul Ecrit

	2	3	
X	1	0	

- L'élève peut donc écrire zéro au rang des unités puisqu'il n'y a pas de bonbon, 3 au rang des dizaines et 2 au rang des centaines dans le calcul écrit et voir que le résultat est le nombre initial décalé d'un rang.

Représentation imagée



Calcul Ecrit

	2	3	
X	1	0	
2	3	0	

D. 4° étape :

♦ **But :**

- Généraliser la multiplication par les multiples de 10.
- Montrer que multiplier par k * 10 revient au même que multiplier par k et décaler le résultat d'un rang vers la gauche.

♦ **Type d'exercice :**

- J'ai X bonbons dans une boîte. Combien ai-je de bonbons si je prends Y boîtes identiques ?
- X quelconque, $Y = k * 10$, $k = 1, 2, ..., 9$.

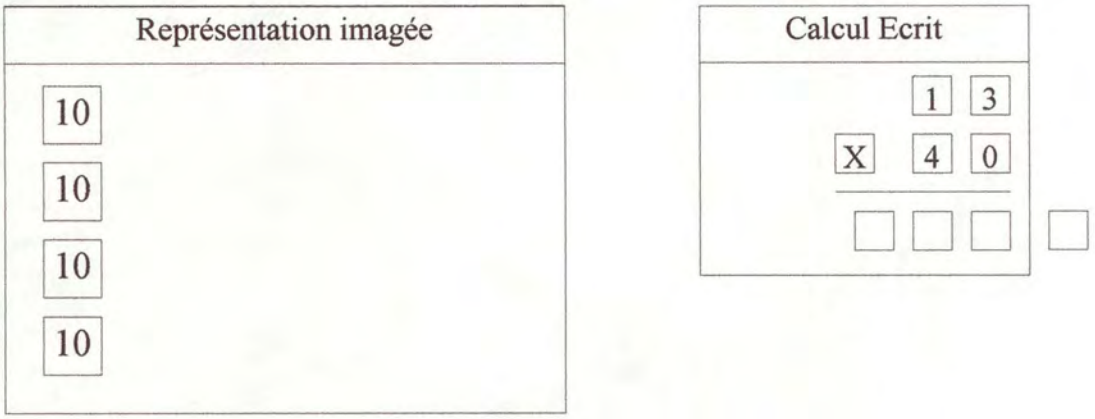
♦ **Exemple :**

- J'ai 13 bonbons dans une boîte. Combien ai-je de bonbons si je prends 40 boîtes identiques ?

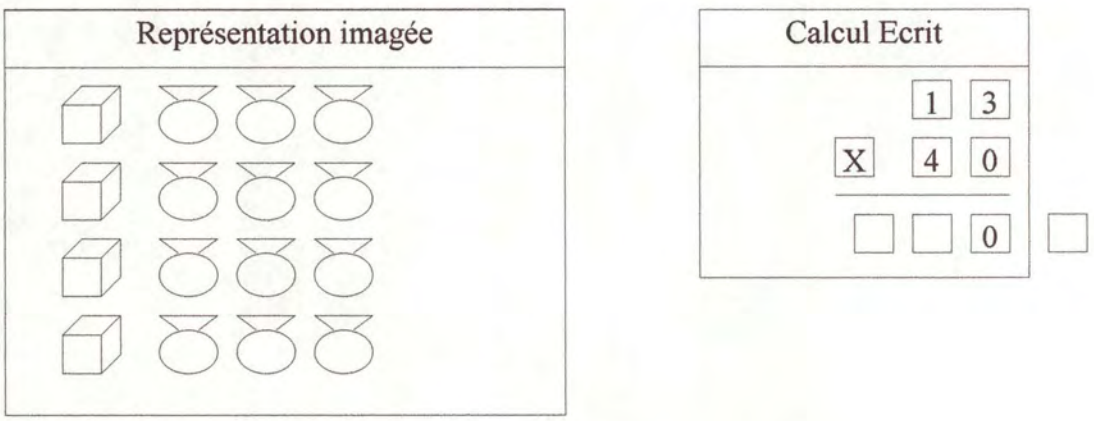
♦ **Déroulement :**

- L'élève sélectionne le multiplicande et le chiffre des dizaines du multiplicateur pour faire apparaître la représentation imagée de la multiplication. Ici, on ne détaille plus la multiplication par 10. On suppose que l'élève connaît le résultat de cette opération. La boîte contenant le nombre 10 représente 10 boîtes identiques.

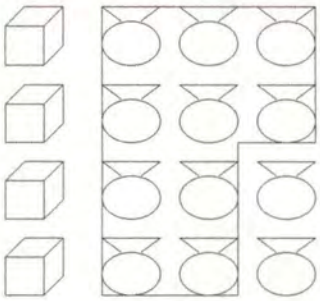
- Multiplier un nombre par k*10 revient au même que multiplier par k puis décaler le résultat d'un rang vers la gauche. Pour comprendre ce principe les élèves doivent avoir assimilé l'étape précédente.



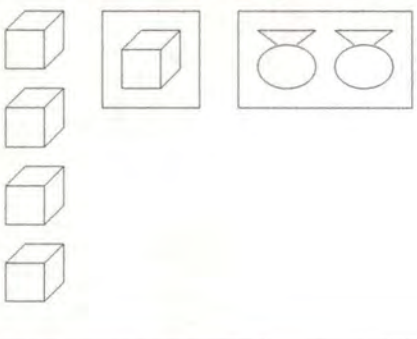
- L'élève peut alors ouvrir les boîtes pour faire apparaître leur contenu, c'est-à-dire 10 fois le multiplicande (c'est ce qu'il a appris à l'étape 3). Il sait déjà que le résultat au rang des unités sera zéro. Il peut donc l'inscrire.



- Quand toutes les boîtes sont ouvertes, l'élève peut regrouper les sachets par groupes de 10 pour obtenir des caisses.

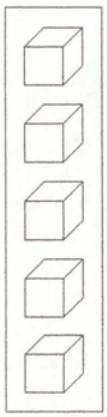
Représentation imagée	Calcul Ecrit
	1 3 X 4 0 0

- Quand il lui reste moins de 10 sachets, il peut inscrire le résultat au rang des dizaines en sélectionnant les sachets restants. Il peut aussi inscrire le report dans la case des reports.

Représentation imagée	Calcul Ecrit
	1 3 X 4 0 2 0 1

- Il lui reste enfin a inscrire le résultat au rang des centaines en ajoutant la caisse report aux autres caisses.

Représentation imagée



Calcul Ecrit

		1	3
X		4	0
5	2	0	

E. 5° étape :

♦ **But :**

- Généraliser la multiplication de 2 nombres quelconques.
- Bien montrer qu'elle se décompose en une addition de 2 multiplications d'un nombre par un chiffre. Ces deux multiplications plus simples ont été détaillées aux étapes précédentes.

♦ **Type d'exercice :**

- J'ai X bonbons dans une boîte. Combien ai-je de bonbons si je prends Y boîtes identiques ?
- $X < 100$, $Y < 100$.

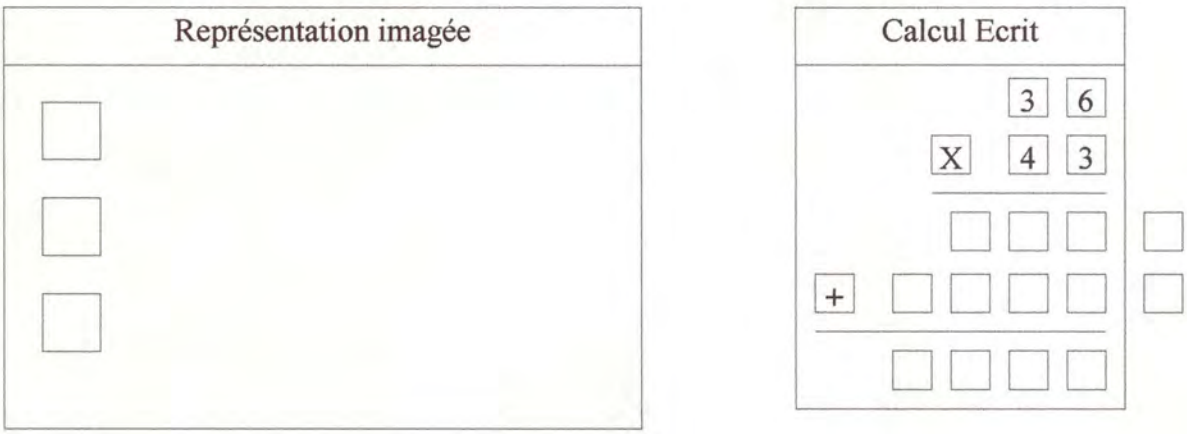
♦ **Exemple :**

- J'ai 36 bonbons dans une boîte. Combien ai-je de bonbons si je prends 43 boîtes identiques ?

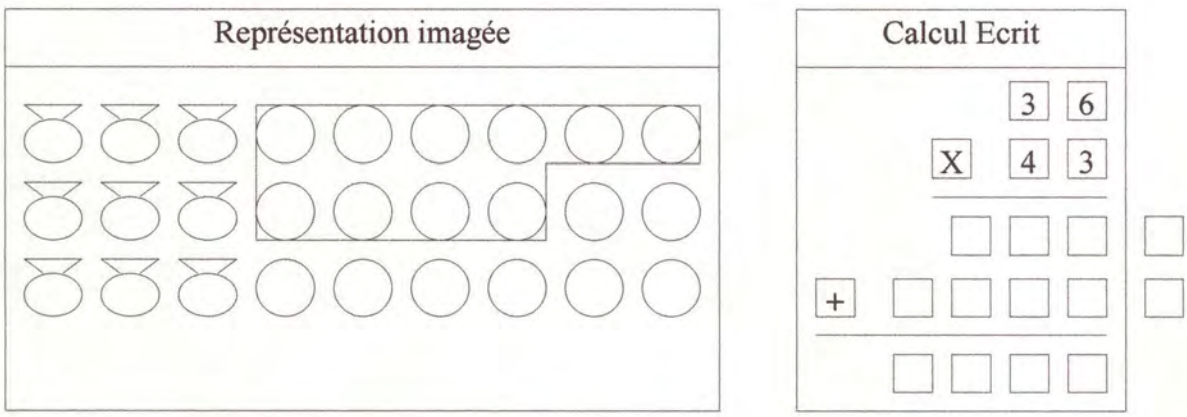
♦ **Déroulement :**

- Le problème qui peut se poser lorsque les élèves doivent multiplier deux nombres d'au moins deux chiffres est celui de la découpe de cette multiplication en addition de plusieurs multiplications plus simples. Les élèves doivent donc faire cette découpe d'une certaine manière. Dans l'exemple ci-dessous, ils doivent d'abord multiplier 36 par le chiffre des unités du multiplicateur puis par le chiffre des dizaines du multiplicateur.

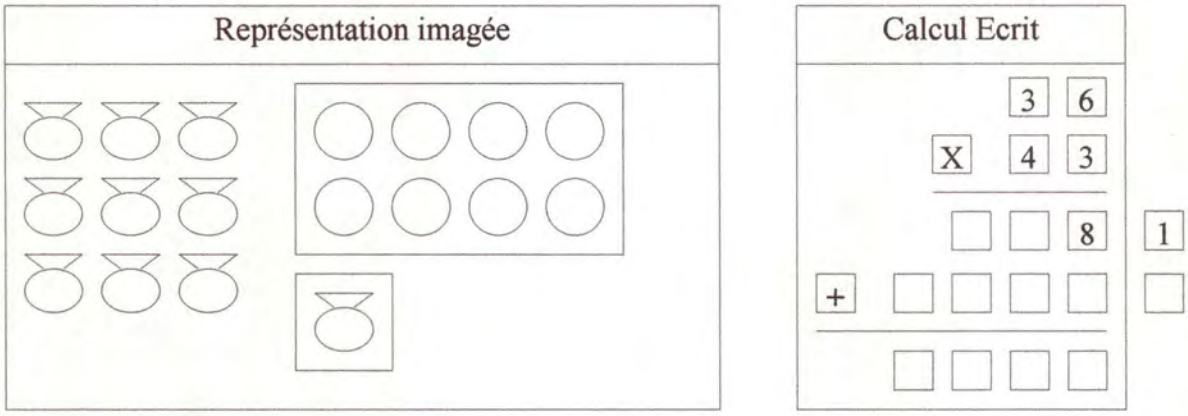
- L'élève doit sélectionner le chiffre des unités du multiplicateur et le multiplicande pour faire apparaître la représentation imagée de cette opération. Il doit donc sélectionner le chiffre 3 et le nombre 36. Il voit alors apparaître 3 boîtes.



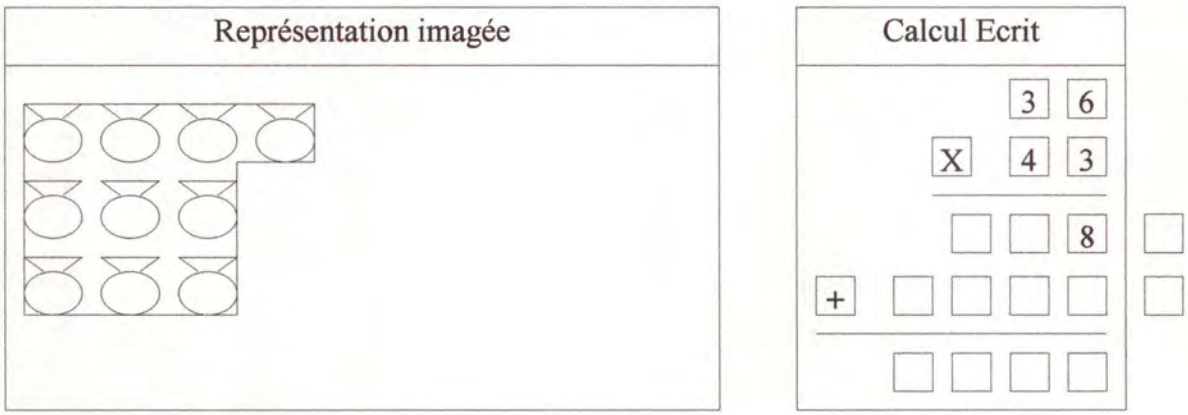
- Il peut alors ouvrir les 3 boîtes pour voir ce qu'elles contiennent (36 bonbons chacune) et regrouper les bonbons pour obtenir des sachets reports.



- Quand il reste moins de 10 bonbons, il peut inscrire le résultat au rang des unités et le report dans la case des reports en sélectionnant les objets correspondants.



- Il peut ensuite regrouper les sachets au maximum, sans oublier le sachet report, pour obtenir le résultat au rang des dizaines et le report des centaines.



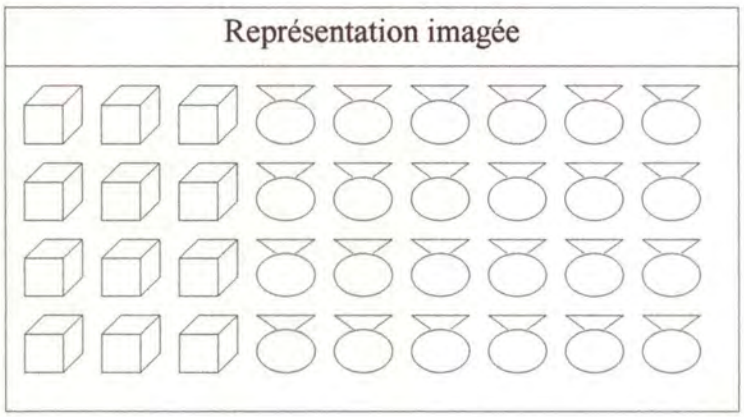
- Ici, il obtient 1 caisse et il ne reste aucun sachets. Il peut donc inscrire les résultats aux rang des dizaines et des centaines.

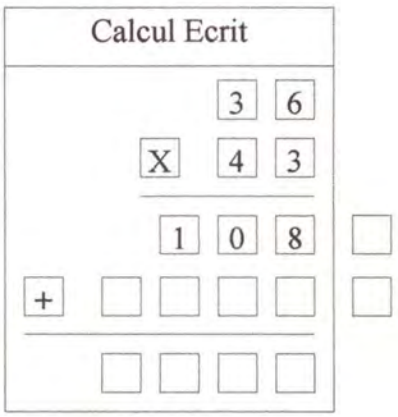
Représentation imagée	Calcul Ecrit
	3 6 X 4 3 1 0 8 +

- L'élève peut maintenant passer à la deuxième partie de la multiplication. Il doit sélectionner le chiffre des dizaines du multiplicateur et le multiplicande pour faire apparaître la représentation imagée de l'opération. Il voit apparaître 4 boîtes de 10 qui contiennent chacune 10 fois le multiplicande.

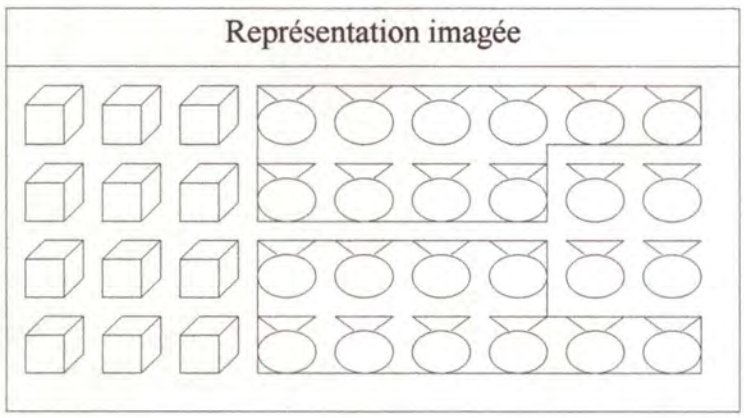
Représentation imagée	Calcul Ecrit
10 10 10 10	3 6 X 4 3 1 0 8 +

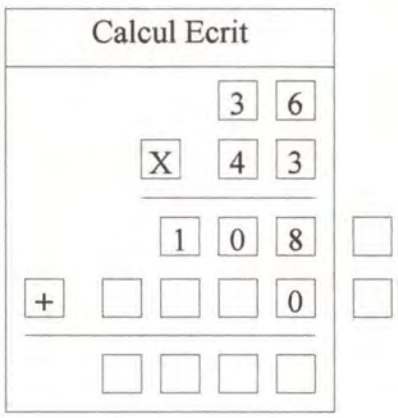
- Il ouvre alors les boîtes pour faire apparaître leur contenu.

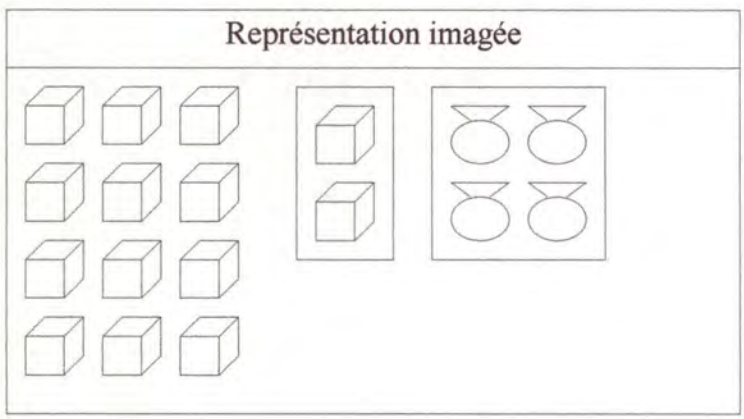


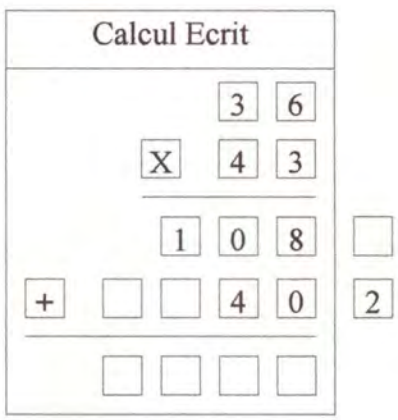


- Quand toutes les boîtes sont ouvertes, il doit regrouper les sachets au maximum pour obtenir le résultat au rang des dizaines et le report qu'il peut inscrire dans le calcul écrit. Puisqu'il n'y a pas de bonbon, il peut inscrire 0 au rang des unités.

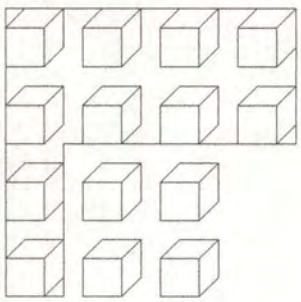


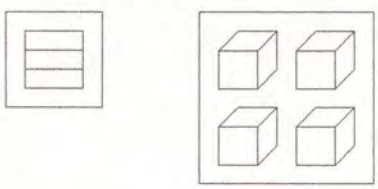




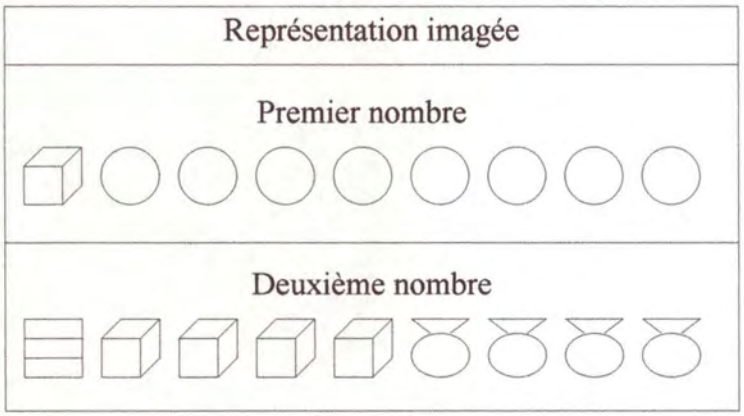


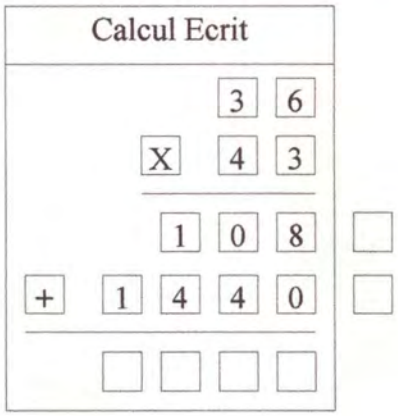
- Il fait de même avec les caisses, sans oublier d'y ajouter les 2 caisses reports, pour obtenir le résultat au rang des centaines et au rang des milliers.

Représentation imagée	Calcul Ecrit
	3 6 X 4 3 1 0 8 □ + □ □ 4 0 □ □ □ □ □

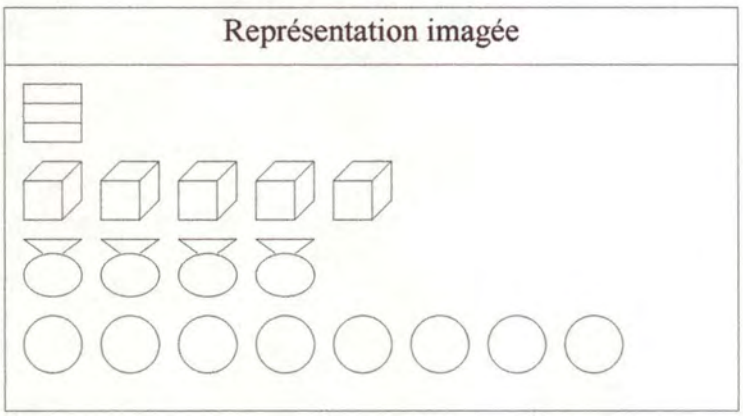
Représentation imagée	Calcul Ecrit
	3 6 X 4 3 1 0 8 □ + 1 4 4 0 □ □ □ □ □

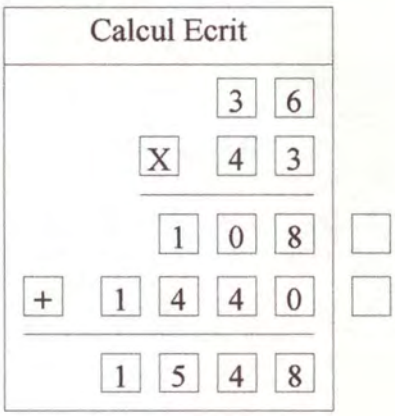
- Il ne lui reste plus qu'à effectuer une addition avec les deux résultats qu'il a obtenus en utilisant la méthode de regroupement qui lui a déjà été enseignée dans le travail sur l'addition.





- Dans cette addition, il n'y a aucun regroupement possible. Le résultat est donc immédiat.





2.2.2. L'apprentissage de la division :

De même que pour la multiplication, j'ai relevé plusieurs étapes que les élèves doivent franchir pour parvenir à effectuer une division de deux nombres quelconques. Cette découpe n'est pas unique. Il est tout à fait possible de trouver une autre méthode d'apprentissage de la division.

La première étape est la compréhension par l'élève de la division et l'explication du principe de la distribution. Quand ce principe est acquis, il faut passer à des exercices plus compliqués faisant intervenir des nombres plus grands. D'abord les dizaines qu'ils doivent transformer en unités quand il n'y en a plus assez pour les distribuer, puis les centaines et les milliers. Dans ces trois étapes, nous envisageons la division avec reste. Les exercices se terminent donc lorsque le reste est inférieur au diviseur.

Je donnerai, pour chacune des 3 étapes dégagées, son but, le type d'exercice proposé aux élèves, un exemple concret et son déroulement possible. Je tenterai également de montrer les problèmes qui peuvent se poser aux élèves qui apprennent la division.

A. 1° étape :

♦ **But :**

- Faire comprendre ce qu'est une division (avec reste différent de zéro).
- Montrer l'usage du zéro dans la division.

♦ **Exercice :**

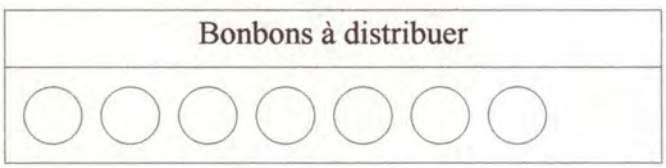
- J'ai X bonbons et je voudrais les distribuer équitablement dans Y boîtes. Combien y en aura-t-il dans chaque boîte et combien en restera-t-il ?
- $X < 10$, $Y < 10$, $Y < X$.

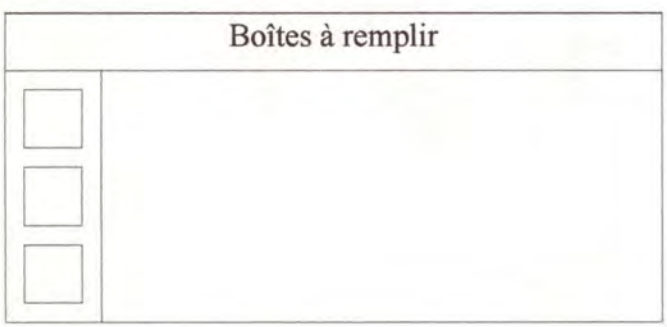
♦ **Exemple :**

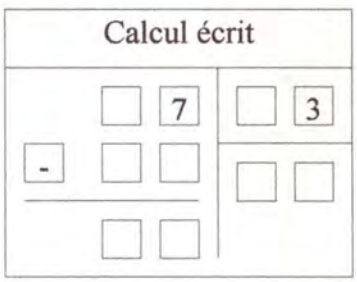
- J'ai 7 bonbons et je voudrais les distribuer équitablement dans 3 boîtes. Combien y en aura-t-il dans chaque boîte et combien en restera-t-il ?

♦ **Déroulement :**

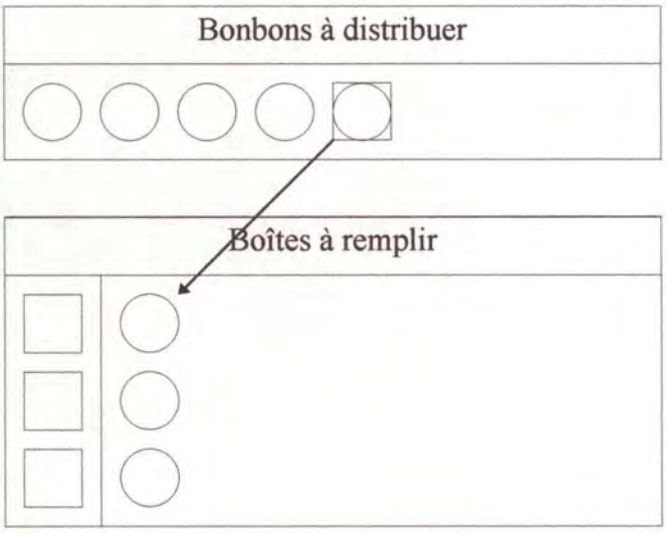
- L'élève a devant lui une série de bonbons qu'il doit distribuer dans plusieurs boîtes. Il peut soit distribuer les bonbons un à un dans les trois boîtes, soit il en prend 2 à la fois. Il ne faut, en effet, arrêter les enfants que quand il commettent une erreur.

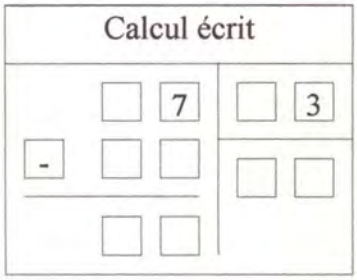






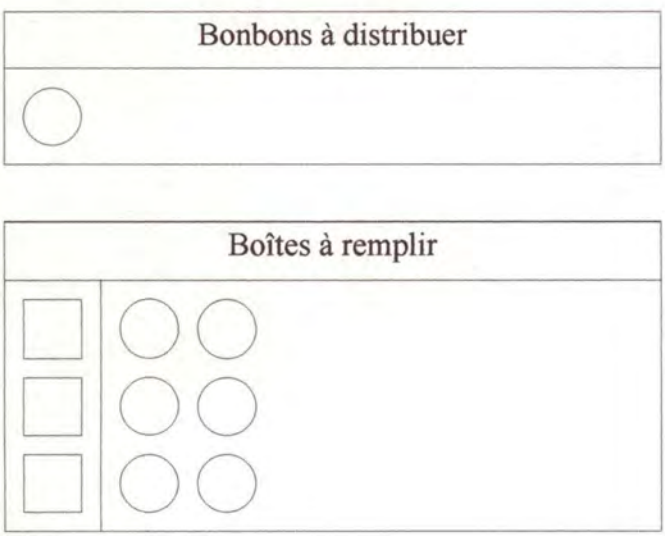
- Supposons que l'élève sélectionne un bonbon à la fois. Il en met un dans chaque boîte.

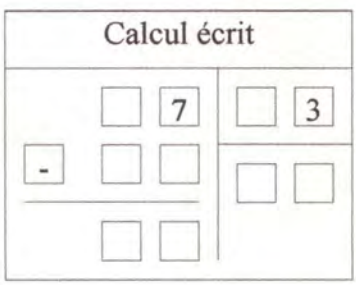




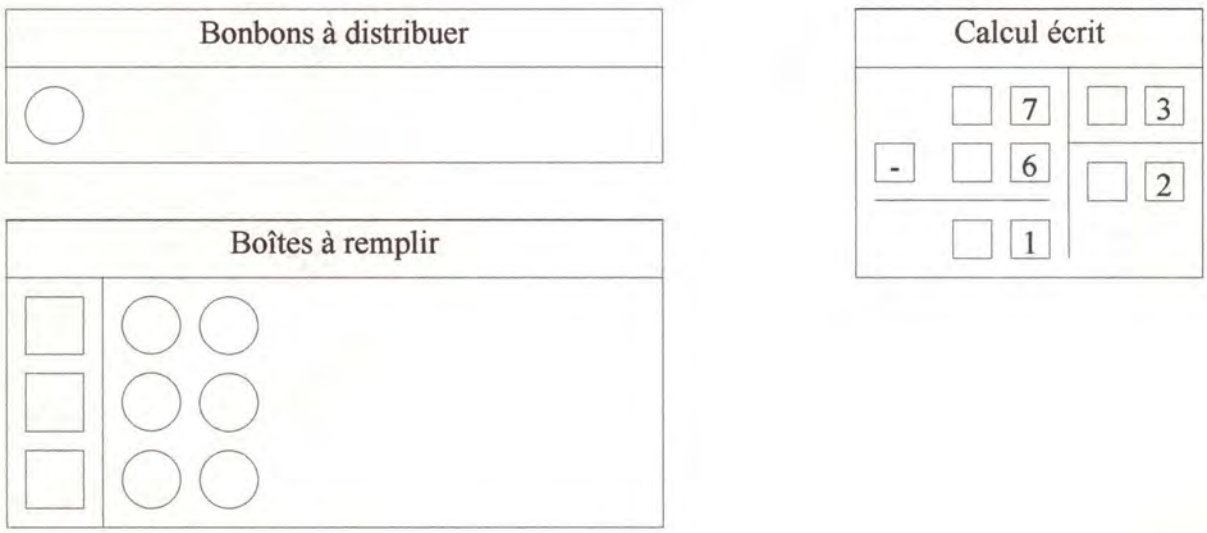
- Il répète cette opération avec trois autres bonbons. Il remarque alors qu'il n'y en a plus assez pour en mettre encore un dans chaque boîte.

- L'élève peut avoir des problèmes à cet endroit de la division s'il tente de distribuer le bonbon restant. Il faut dès lors bien lui expliquer qu'il faut arrêter la distribution quand le nombre d'éléments restants est inférieur au diviseur. Sinon, une des boîte contiendra plus de bonbons que les autres et la distribution n'est plus équitable.





- Il peut inscrire le résultat de la division, c'est-à-dire 2, dans la case réservée à celui-ci dans le calcul écrit. Il doit ensuite soustraire les 6 (c'est-à-dire le nombre de bonbons qu'il a placé dans les boîtes) au dividende, c'est-à-dire 7. Le résultat qu'il obtient est 1 (c'est-à-dire le nombre de bonbons qu'il reste).



- Il faut faire remarquer à l'élève que la division est l'inverse de la multiplication. Si on multiplie le résultat par le diviseur (2 fois 3) et qu'on y ajoute le reste (1), on obtient 7, c'est-à-dire le dividende. Pour effectuer une division, il doit regarder combien de fois le diviseur va dans le dividende, jusqu'à ce qu'il obtienne un reste inférieur au diviseur.

- Pour montrer l'effet du zéro dans la division, on fait le même exercice avec :
  - soit 0 bonbon à distribuer ($X = 0$, $Y > 0$) : l'élève voit qu'il n'y a rien à distribuer et qu'il n'y aura rien dans les boîtes et donc pas de reste.
  - soit X bonbons à distribuer dans 0 boîtes ($X > 0$, $Y = 0$) : l'élève voit que ce n'est pas possible de distribuer des bonbons qui n'existent pas, la division par zéro n'est donc pas possible.

B. 2° étape :

♦ **But :**

- Montrer que le principe de distribution est le même quand il a des dizaines dans le dividende.
- Montrer que quand il n'y en a plus assez, il faut déballer les sachets pour obtenir 10 bonbons et distribuer ceux-ci.

♦ **Exercice :**

- J'ai X bonbons et je voudrais les distribuer équitablement dans Y boîtes. Combien y en aura-t-il dans chaque boîte et combien en restera-t-il ?
- $X > 10$, $Y < 10$.

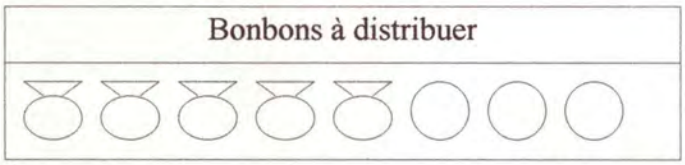
♦ **Exemple :**

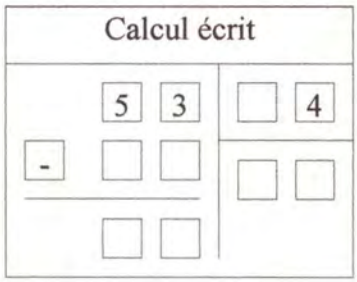
- J'ai 53 bonbons et je voudrais les distribuer équitablement dans 4 boîtes. Combien y aura-t-il de bonbons dans chaque boîte et combien en restera-t-il ?

♦ **Déroulement :**

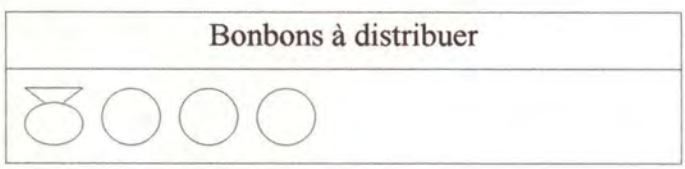
- Dans cette étape, les élèves peuvent avoir des difficultés à comprendre pourquoi il faut commencer par distribuer les dizaines avant de distribuer les unités. Avec la représentation imagée de la division, l'élève pourra voir pourquoi il doit procéder ainsi.

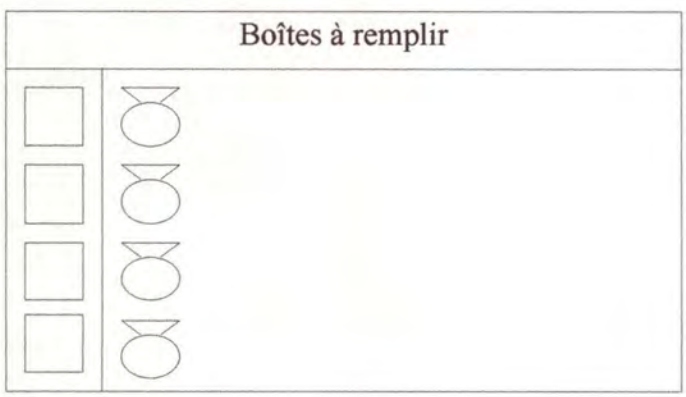
- L'élève a 5 sachets et 3 bonbons qu'il doit distribuer équitablement dans 4 boîtes.

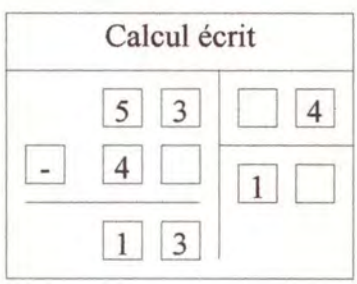




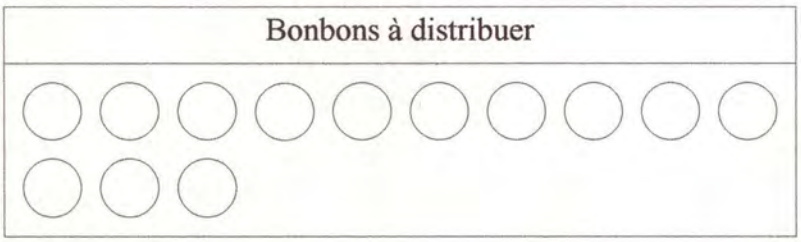
- Il peut mettre un sachet dans chaque boîte et inscrire 1 au résultat puisqu'il y a une dizaine dans chaque boîte. Il doit aussi soustraire les 4 sachets qu'il vient de déplacer (puisque les sachets représentent les dizaines, il soustrait 4 au rang des dizaines dans le calcul écrit). Il lui en reste 1 qu'il ne peut placer nulle part sinon la distribution n'est plus équitable.

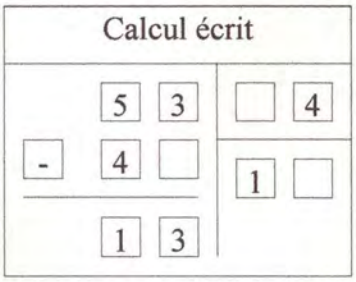




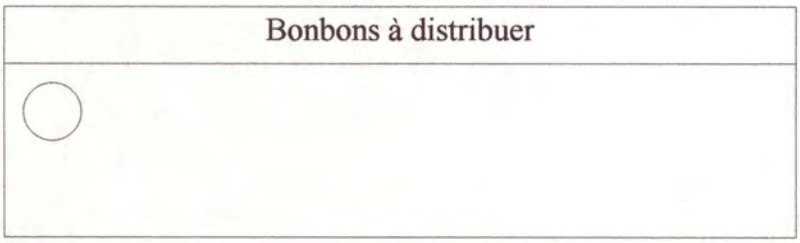


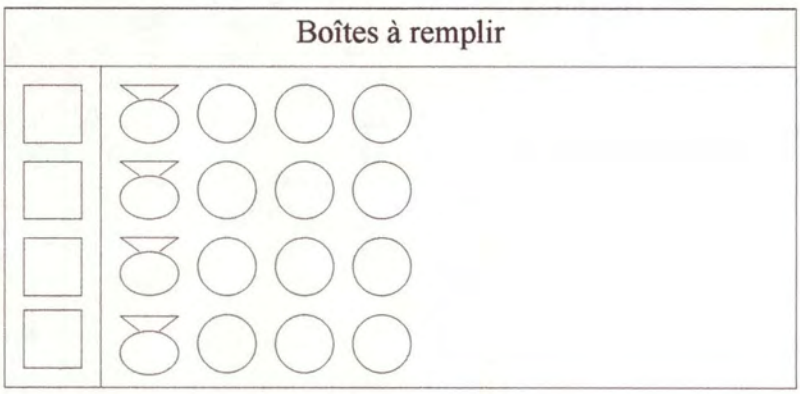
- Il lui reste donc un sachet et 3 bonbons. Il peut ouvrir le sachet pour obtenir un total de 13 bonbons qu'il pourra distribuer dans les boîtes jusqu'à ce qu'il n'y en ait plus assez .

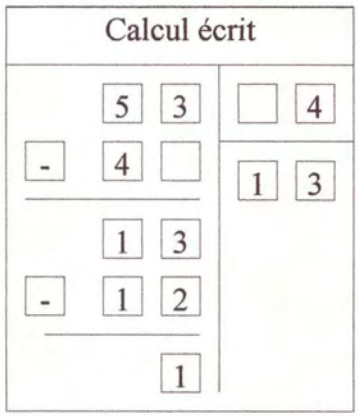




- Quand il a ouvert le sachet et distribué au maximum les bonbons, il peut inscrire le résultat dans le calcul écrit (c'est-à-dire 3 car il y a 3 bonbons dans chaque boîte) et soustraire les bonbons qu'il vient de déplacer au dividende. Il lui reste donc un bonbon qu'il ne peut mettre nulle part.


Bonbons à distribuer


Boîtes à remplir


Calcul écrit
5 3
- 4
1 3
- 1 2
1
4
1 3

C. 3° étape :

♦ **But :**

- Généraliser la division aux centaines et aux milliers.

♦ **Exercice :**

- J'ai X bonbons et je voudrais les distribuer équitablement dans Y boîtes. Combien y en aura-t-il dans chaque boîte et combien en restera-t-il ?
- X et Y quelconques, $Y < X$.

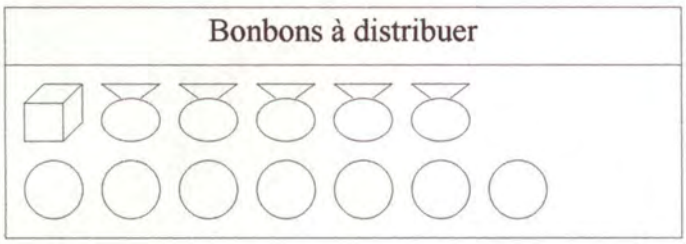
♦ **Exemple :**

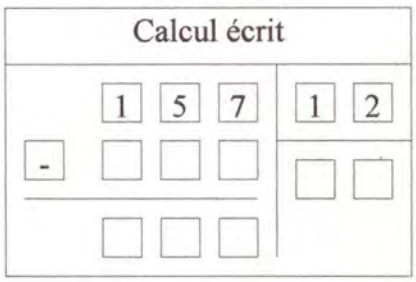
- J'ai 157 bonbons et je voudrais les distribuer équitablement dans 12 boîtes. Combien y en aura-t-il dans chaque boîte et combien en restera-t-il ?

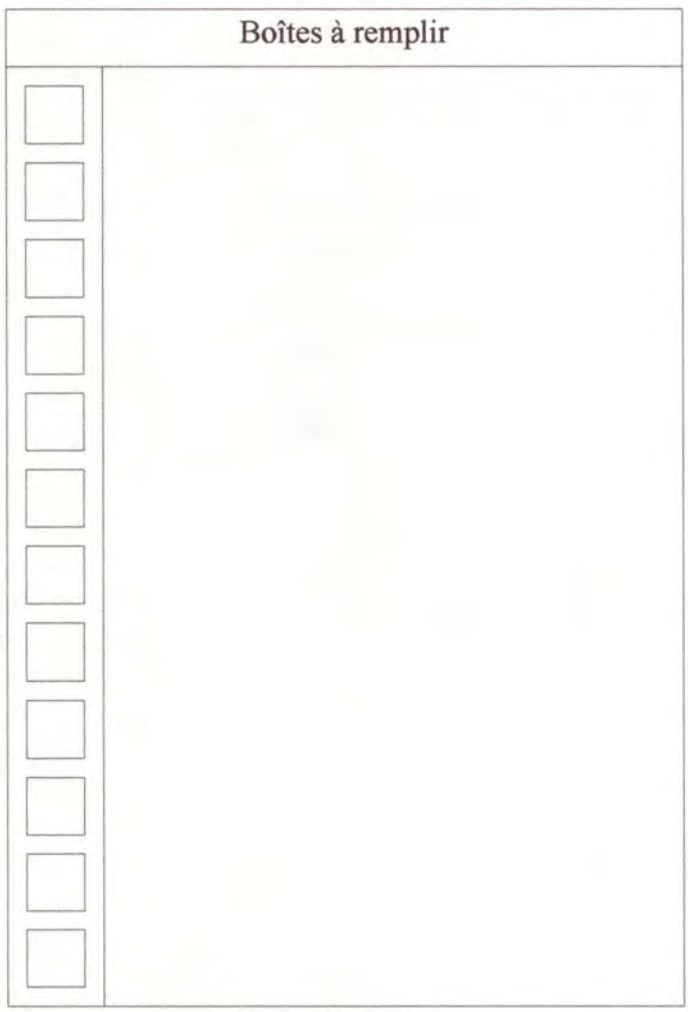
♦ **Déroulement :**

- Cet exercice doit montrer aux élèves qu'il est plus facile d'effectuer le calcul écrit d'une division que de l'effectuer avec la représentation imagée. En effet, quand le diviseur devient grand, cela devient difficile de dessiner toutes les boîtes.

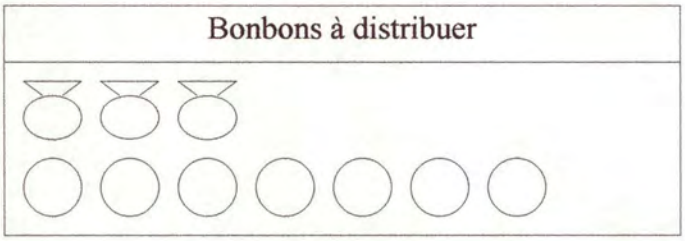
- L'élève a devant lui une caisse, 5 sachets et 7 bonbons. Il doit les distribuer dans les 12 boîtes en commençant par les caisses.

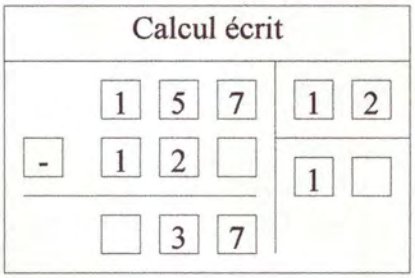


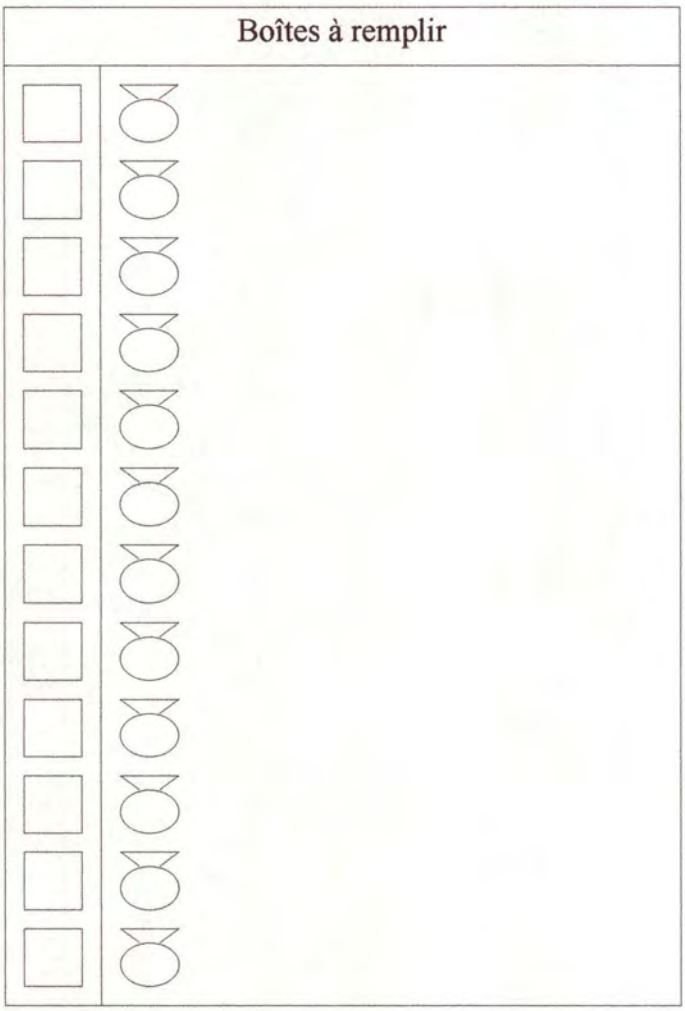




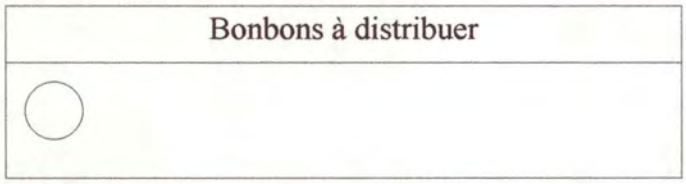
- Il n'a pas assez de caisses pour en mettre une dans chaque boîte. Il doit donc ouvrir la caisse pour obtenir des sachets et distribuer ceux-ci au maximum dans les boîtes. Puisqu'il en a mis 1 dans chaque boîte, il peut inscrire 1 dans la case du résultat au rang des dizaines (puisqu'il a distribué des sachets) et soustraire 12 sachets au dividende dans le calcul écrit.

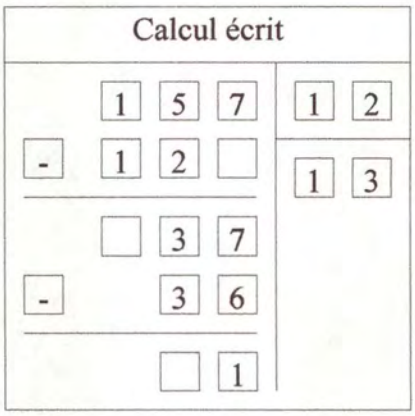


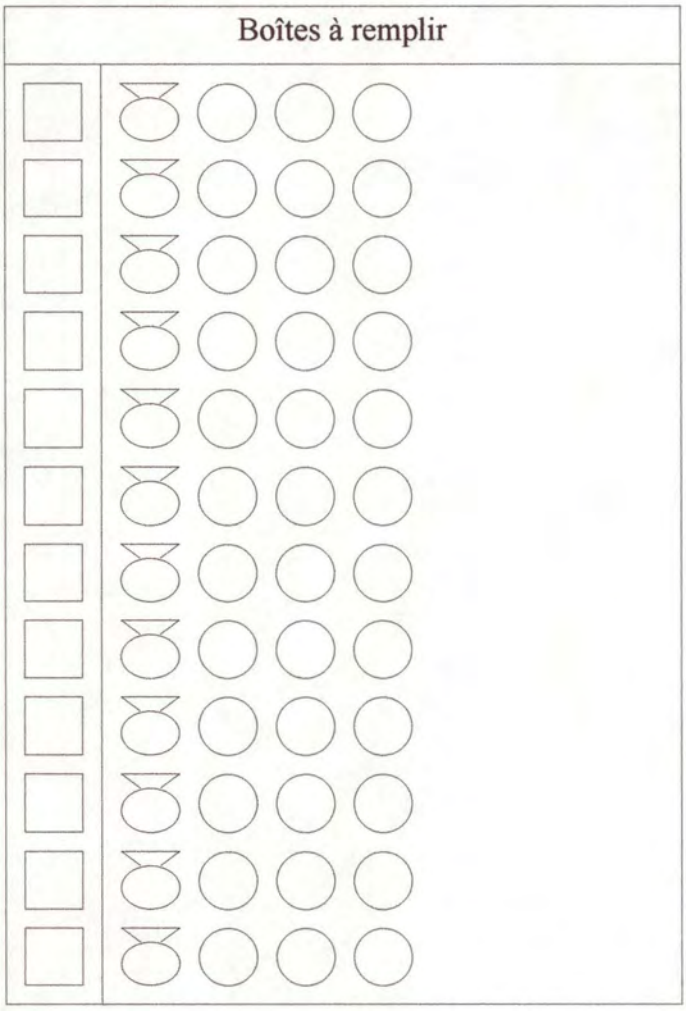




- L'élève peut maintenant répéter l'opération pour les sachets. Il ouvre tous les sachets pour obtenir des bonbons. Il en a 37 qu'il peut distribuer au maximum dans les boîtes. Il sait en mettre 3 dans chaque boîte. Il peut ensuite inscrire le résultat au rang des unités et soustraire les 36 bonbons qu'il vient de distribuer pour obtenir le reste.







- Il lui reste un bonbon qu'il ne peut mettre nulle part.

2.3. Les éléments nouveaux.

2.3.1. Les objets de base.

Les objets de base que j'ai définis sont différents de ceux décrits dans le travail sur l'addition et la soustraction. En effet, les icônes représentants les « bonbons », les « sachets », les « caisses », et les « armoires » ainsi que les icônes représentants les chiffres du calcul écrit sont définis comme des objets informatiques (voir point 3.2.1.).

L'avantage de cette méthode est que les objets « connaissent » les actions que l'on peut effectuer ou non sur eux. Par exemple, un objet « bonbon » sait qu'il ne peut pas être décomposé en 10 objets de rang directement inférieur puisque nous travaillons avec des nombres naturels. Cette propriété est inscrite dans sa définition. Par contre, un objet « caisse » sait qu'il peut être « ouvert » et remplacé par 10 objets « sachet ». Cette action d'ouverture est prévue dans sa définition.

Cette méthode permet de mieux traduire la réalité. En effet, si l'on utilise les techniques exposées ci-dessus pour expliquer la multiplication aux enfants avec de vrais bonbons, de vrais sachets, etc., ceux-ci sont des objets réels que les enfants peuvent toucher et avec lesquels ils peuvent jouer. Ils pourront ouvrir un sachet pour en sortir les 10 bonbons qu'il contient mais ils ne pourront pas ouvrir un bonbon car il ne contient rien. C'est cette réalité que j'ai voulu traduire en préférant cette conception des objets à celle décrite dans le travail sur l'addition et la soustraction [Biblio 4].

L'analyse logicielle de ces objets et la description de ceux-ci du point de vue informatique se trouvent au point 3.2.1.

2.3.2. Les modifications de l'addition.

Les objets de base étant différents, j'ai également redéfini les objets d'addition qui les utilisent. Néanmoins, j'ai gardé les principes qui avaient été dégagés lors de l'analyse pédagogique du travail sur l'addition et la soustraction (voir [Biblio 4]).

Le premier objet d'addition que je propose permet de faire la somme de plusieurs chiffres de même rang. Il permet aussi de représenter le calcul écrit de cette opération et son équivalant sous forme imagée.

Le deuxième est un objet qui permet de faire la somme de plusieurs nombres. Cet objet utilise donc le précédent (car une somme de nombres est composée de plusieurs sommes de chiffres de même rang).

Le dernier objet que j'ai défini permet de visionner le déroulement de l'opération d'addition de plusieurs nombres sous forme imagée. Cet objet sera donc utile pour la multiplication. En effet, une multiplication de 2 nombres est une addition de 2 multiplications plus simples.

L'analyse logicielle de ces trois objets se trouve au point 3.2.2.

En plus de ces modifications d'objets, j'ai apporté un changement au processus d'addition tel qu'il est défini dans le travail sur l'addition et la soustraction : l'emballage manuel. Auparavant, l'emballage des objets « bonbon », « sachet », etc. se faisait automatiquement, c'est-à-dire, que lorsque l'élève avait sélectionné 10 objets de même rang, ils étaient directement remplacés par un objet de rang supérieur.

Ce système est intéressant pour les élèves qui connaissent les nombres et le principe du report. Ils savent pourquoi ces 10 objets sont remplacés par un autre. Mais, pour les élèves qui ne le savent pas ou qui débutent dans l'apprentissage de l'addition, il est préférable qu'il ne se produise pas de miracle. C'est pourquoi j'ai introduit l'emballage manuel. Quand l'élève a

sélectionné les objets qu'il veut emballer, il doit enfoncer un bouton d'« emballage » qui provoquera le remplacement des objets sélectionnés par un objet de rang supérieur et cela, uniquement s'ils sont au nombre de 10. S'il y en a plus ou moins de 10, l'emballage ne se fera pas et les élèves seront avertis de leur erreur par l'envoi d'un message qui leur expliquera le pourquoi et le comment du regroupement et du report.

2.3.3. La multiplication.

Comme pour l'addition, les objets de multiplication sont définis de manière à proposer aux élèves deux représentations différentes de l'opération. Les enfants pourront donc effectuer le calcul écrit de la multiplication et jouer avec la représentation imagée de celle-ci. Ils pourront également passer à volonté de la représentation chiffrée à la représentation imagée et inversement. Ainsi, si un élève éprouve des difficultés à résoudre le calcul écrit d'une opération, il pourra facilement passer à la représentation imagée qui l'aidera à comprendre et à y voir plus clair.

Le principe de base que j'ai développé pour expliquer la multiplication aux enfants est celui de la « boîte ». Pour expliquer ce que font 5 fois 3, il suffit de leur montrer 3 « boîtes », chaque « boîte » contenant le multiplicande. Dans ce cas-ci, chaque « boîte » contient 5 « bonbons ». L'ensemble des « boîtes » représentent donc le multiplicateur et leur contenu représente le multiplicande. Faire 5 fois 3 revient donc à prendre 3 « boîtes » qui contiennent chacune 5 « bonbons » et à compter, grâce au principe de regroupement de l'addition, le nombre total de bonbons que l'on a. J'ai choisi cette méthode car, contrairement à l'addition où l'on peut montrer les deux nombres à additionner sous forme de « bonbons », on ne peut pas faire 5 « bonbons » fois 3 « bonbons ». En effet, on obtiendrait des « bonbons » au carré et non le résultat de la multiplication de 5 par 3.

Pour expliquer la multiplication par 10, j'utilise le même principe. L'élève regardera le contenu de 10 « boîtes » identiques et regroupera leur contenu pour obtenir le résultat. Il pourra ainsi remarquer que le résultat sera le multiplicande décalé d'un rang vers la gauche dans le calcul écrit (avec un zéro au rang des unités). Dans la représentation imagée, il aura des « caisses » là où il avait des « sachets » et il aura des « sachets » là où il avait des « bonbons ». Donc en décalant le multiplicande d'un rang vers la gauche au lieu de comptabiliser le contenu de 10 « boîtes », il ira beaucoup plus vite.

Par la suite, on pourra remplacer 10 « boîtes » par une « boîte de 10 ». Ceci permettra de gagner de la place dans la représentation imagée d'une multiplication de deux nombres

quelconques. Les élèves auront donc préalablement appris qu'une « boîte de 10 » contenait 10 fois le multiplicande. Pour représenter 13 fois 40, on utilisera 4 « boîtes de 10 » qui contiendront chacune 10 fois 13.

Un dernier principe qu'il faut expliquer aux enfants qui apprennent à multiplier est l'ordre dans lequel ils doivent effectuer l'opération. Une multiplication de deux nombres supérieurs à 10 se décompose, en effet, en une addition de deux multiplications d'un nombre par un chiffre. Il est préférable d'effectuer d'abord la multiplication du multiplicande par le chiffre des unités du multiplicateur, puis celle du multiplicande par le chiffre des dizaines du multiplicateur. De plus, à l'intérieur de ces deux multiplications plus simples, il est nécessaire de multiplier d'abord les chiffres des rangs les plus petits avant de multiplier les chiffres des rangs plus grands (comme pour l'addition).

Un petit détail qui a son importance : le report. Dans une multiplication, on peut trouver des reports. Ceux-ci peuvent être inscrits pour ne pas être oubliés. Mais par la suite, dans le but de gagner du temps et d'être plus efficace, les élèves pourront retenir les reports mentalement.

Les objets de multiplication qui ont été définis sur les principes décrits ci-dessus seront exposés au point 3.2.3.

2.3.4. La division.

Le principe de base que j'ai développé pour l'apprentissage de la division est la distribution des « bonbons » dans des « boîtes ». C'est le principe inverse de la multiplication. En effet, multiplier revenait à compter les bonbons qu'il y avait dans les « boîtes », chaque boîte contenant le même nombre de bonbons. La division consiste à distribuer les « bonbons » dans les « boîtes » équitablement. A la fin de la division, il doit y avoir le même nombre de bonbons dans chaque boîte.

L'opération de division se base donc sur la multiplication mais aussi sur la soustraction. En effet, pour connaître le résultat de 7 divisé par 3, il faut regarder combien de fois va 3 dans 7 (ce qui fait 2) puis soustraire au dividende le résultat de 2 fois 3 pour obtenir le reste. Pour effectuer une division efficacement, il faut connaître parfaitement la soustraction et la multiplication. L'apprentissage de cette opération vient donc logiquement après l'apprentissage des trois autres opérations de base.

Comme pour les opérations précédentes, la division peut se faire soit sous forme imagée (c'est-à-dire qu'il faut remplir des « boîtes » avec des « bonbons »), soit sous forme de calcul écrit. Ainsi, au début de l'apprentissage, la représentation imagée permet de bien présenter le principe de la division, et, par la suite, la résolution du calcul écrit permet d'effectuer la division plus rapidement.

2.3.5. Le gestionnaire d'événements.

Lors de l'utilisation du logiciel, l'interaction entre la machine et l'élève définit des événements. Par exemple, la tentative d'ouverture d'un « bonbon » ou d'un « sachet » est un événement. Lorsqu'un tel événement se produit, il faut pouvoir y réagir. C'est la tâche du gestionnaire d'événements que je vais exposer ci-après.

Ce gestionnaire, pour accomplir la tâche qui lui est fixée, c'est-à-dire, répondre aux événements, doit tenir compte de l'état d'avancement de l'opération en cours. En effet, le gestionnaire d'événements ne doit pas réagir de la même façon à la tentative d'ouverture d'un sachet suivant que le rang courant est le « sachet » ou le « bonbon ».

Le gestionnaire doit aussi tenir compte du niveau des connaissances des élèves. C'est un point important car la connaissance des élèves évolue avec les exercices qu'ils font et le niveau des connaissances est différent en fonction de l'élève. Mais tenir compte de l'évolution de ce niveau de connaissance n'est pas chose facile. Une façon de détourner cette difficulté est de tenir compte de la fréquence de production des événements. Plus une erreur se produit et plus l'élève a des difficultés, donc, plus il est urgent de l'aider en réagissant à son erreur. Par la suite, on pourrait envisager un système de mémorisation des événements de chaque élève et déterminer grâce à cela le niveau de ses connaissances.

Le gestionnaire d'événements doit être construit de manière à pouvoir répondre aux événements en tenant compte des points cités ci-dessus. Il vérifiera donc si l'élève fait ce qu'il faut, quand il le faut. Ce gestionnaire, sans vouloir faire de notre logiciel un générateur de didacticiel, est un premier pas dans cette direction. Il permet en effet une paramétrisation plus fine que celle qui consiste uniquement à définir un nombre maximum d'erreurs avant de réagir à celles-ci. Le gestionnaire permet à un « non programmeur », comme l'enseignant, de déterminer à l'avance le dialogue entre la machine et l'élève. C'est la première étape vers le générateur de didacticiel. Celui-ci pourra être amélioré pour finalement arriver au générateur.

La description du gestionnaire d'événements du point de vue informatique se trouve au point 3.3.

2.3.5.1. Les messages.

Pour réagir aux événements, le gestionnaire pourra, dans certains cas, envoyer des messages aux élèves. Un message est un petit texte qui expliquera éventuellement aux élèves comment ils doivent utiliser le logiciel, comment ils doivent s'y prendre pour faire une multiplication ou pourquoi ils ont fait une erreur. Il y a donc plusieurs types de messages-réponses en fonction de la nature du contenu de ceux-ci. J'ai dégagé cinq types de messages possibles. Cette liste n'est évidemment pas exhaustive et peut être complétée :

- les messages d'aide à l'utilisation du didacticiel (si les élèves ont des difficultés à se servir du logiciel),
- les messages d'explication de l'opération en cours (instructions sur le déroulement de l'opération),
- les messages d'encouragement (en cas de bonne réponse),
- les messages de mise en garde (si l'élève tente de faire une action qui ne convient pas),
- et les messages d'erreurs (si l'élève a fait une action qu'il ne faut pas).

Ces cinq types de messages ne doivent pas être présentés de la même manière aux élèves. C'est pourquoi je les ai distingués.

Les messages d'aide à l'utilisation du didacticiel doivent être visibles en permanence pour que les élèves puissent les consulter quand ils en ont besoin. Ces messages surviennent suite à une erreur de manipulation du didacticiel ou pour aider les élèves à progresser dans leur opération. Il en est de même pour les messages d'explication de l'opération en cours qui servent à montrer la marche à suivre pour effectuer l'opération en question.

Par contre, les messages d'encouragement, les messages de mise en garde et les messages d'erreur ne doivent apparaître que lorsque le professeur le juge nécessaire. Il doit en plus frapper l'attention des élèves, c'est pourquoi il est préférable qu'ils ne soient pas tout le temps visibles (l'élève n'y ferait plus attention). Ces messages apparaîtront donc au-dessus de tout et les élèves devront les acquitter pour être sûr qu'ils les ont vu.

Les enseignants pourront choisir le type de leurs messages en fonction de l'affichage qu'ils veulent donner aux élèves et en fonction de la nature de l'événement auquel ils veulent répondre.

2.3.5.2. Les événements.

Pour permettre aux enseignants qui utiliseront ce gestionnaire de choisir les messages qu'ils veulent donner à leurs élèves, il faut qu'ils connaissent les événements qui peuvent se produire lors du déroulement d'une opération.

La liste des événements qui peuvent se produire lors d'une multiplication et qui est actuellement prévue est exposée ci-dessous. Un programmeur peut en effet rajouter facilement des événements à cette liste. Il lui suffit d'envoyer le message voulu (qui est généré dans les méthodes du logiciel) au gestionnaire d'événements pour que celui-ci puisse y répondre.

Les événements actuellement prévus sont les suivants :

- essai d'emballer un nombre d'éléments différent de 10,
- emballage de 10 éléments,
- essai de déballer un « bonbon »,
- déballage d'un objet qui peut être décomposé,
- mauvaise réponse à l'addition de chiffres,
- mauvaise réponse à la multiplication de chiffres,
- mauvaise réponse au report de l'addition,
- mauvaise réponse au report de la multiplication,
- addition de chiffres,
- multiplication de chiffres.

2.3.5.3. Les états.

Pendant l'exécution de l'opération, le logiciel passe par différents états successifs. Ceux-ci déterminent l'état d'avancement du programme et donc de l'opération en cours (par exemple, le rang courant de l'opération). Combinés aux événements, les états décrivent une situation précise à laquelle les enseignants pourront choisir de répondre ou non. La liste des états possibles actuellement prévus et que l'on peut trouver dans le déroulement de la multiplication est la suivante :

- les chiffres à multiplier sont sélectionnés,
- les chiffres à additionner sont sélectionnés,
- la multiplication a donné 2 chiffres de résultat,
- la multiplication a donné 1 chiffre de résultat,
- l'addition a donné 2 chiffres de résultat,
- l'addition a donné 1 chiffre de résultat,
- le rang courant est l'unité,
- le rang courant est la dizaine,
- le rang courant est la centaine,
- le rang courant est le millier,
- la multiplication est terminée,
- l'addition est terminée.

Ces états, combinés aux événements cités plus haut, forment un couple événement-état. Les enseignants pourront choisir les messages-réponses qu'ils veulent afficher en fonction de ces couples. Pour modifier un message déjà existant ou ajouter un message, ils utiliseront un éditeur de messages (voir 2.3.5.5.).

2.3.5.4. La fréquence de production des événements.

Pour pallier à la difficulté de déterminer le niveau des connaissances des élèves, j'ai introduit la notion de fréquence de production des événements. Les événements qui se produiront seront comptabilisés et plutôt que de répondre chaque fois à chaque événement, les enseignants pourront décider de ne répondre à ceux-ci qu'après un certain nombre d'apparitions. Le nombre d'apparitions de l'événement à la suite duquel on doit répondre est appelé la « limite ». Cette « limite » sera choisie par les enseignants et pourra être changée à volonté.

Ainsi, l'enseignant choisira une limite élevée pour des élèves de niveau élevé et il choisira une limite peu élevée pour des élèves de niveau faible. En effet, si la limite d'un événement est de un, après une apparition de cet événement, le gestionnaire y répondra. Les élèves qui ont des difficultés ou qui ne s'en sortent pas ne devront pas commettre plusieurs fois la même erreur avant que le gestionnaire leur explique comment il faut faire.

Si par contre on fixe la limite à trois ou quatre, le gestionnaire ne répondra pas tout de suite et les élèves expérimentés qui commettent une erreur par distraction ne seront pas interrompus sans arrêt. De plus, si on donne tout le temps des explications à ces élèves, ils finiront par ne plus y faire attention. Les enseignants doivent donc bien déterminer ces limites en fonction des élèves qui utilisent le didacticiel.

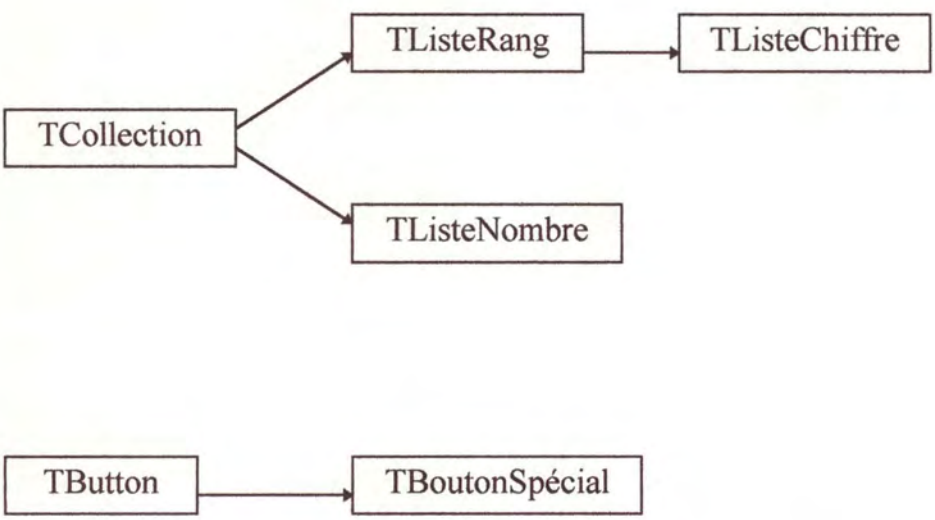
2.3.5.5. L'éditeur de messages.

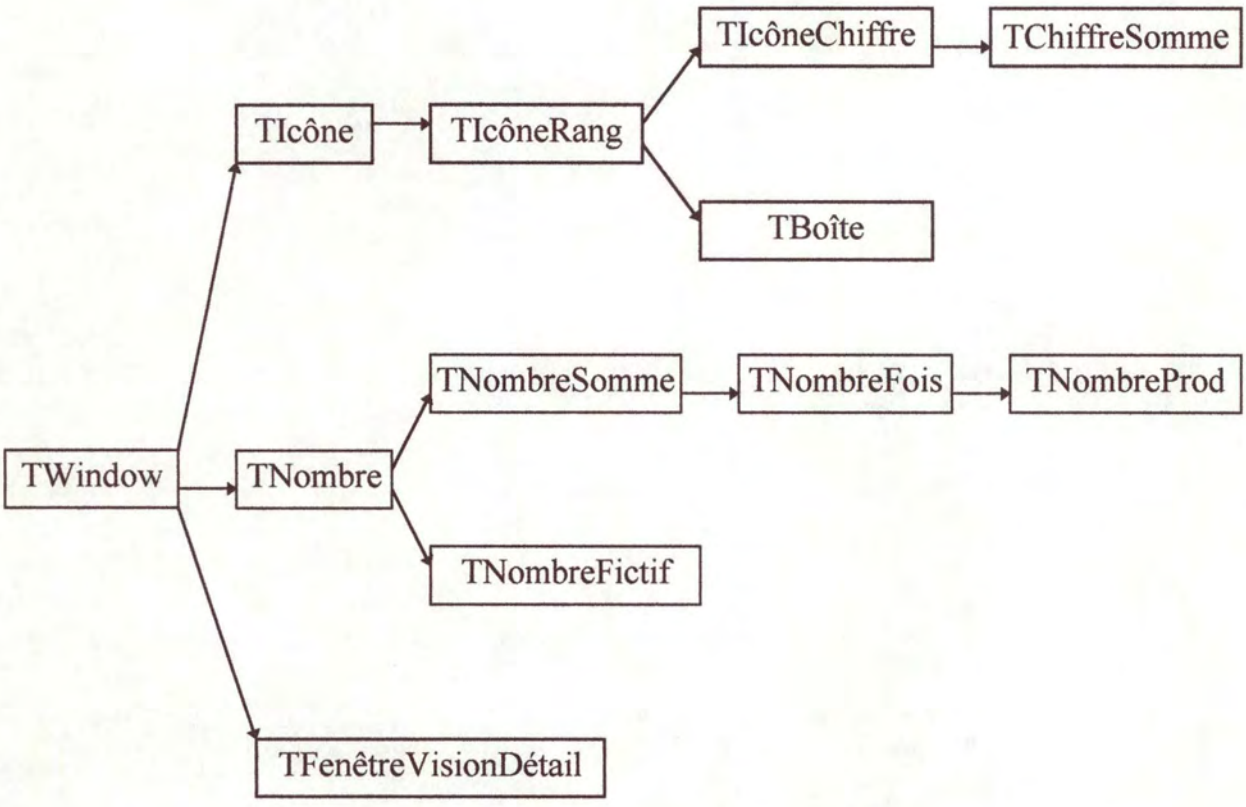
L'éditeur de messages sert à faciliter la modification, l'ajout ou la suppression de messages. Il évite aux enseignants de devoir déchiffrer le code du programme pour changer un message ou la valeur d'une limite. Les enseignants doivent simplement donner le couple événement-état qui correspond au message pour accéder au message et le modifier.

3

ANALYSE LOGICIELLE

3.1. Hiérarchie des objets.




TIcôneChiffre
TChiffreSomme
TIcône
TIcôneRang
TBoîte
TNombreSomme
TNombreFois
TNombreProd
TWindow
TNombre
TNombreFictif
TFenêtreVisionDétail

3.2. Description des objets.

3.2.1. Les objets de base.

3.2.1.1. Objet TIcône :

♦ **Caractéristiques :**

- Objet enfant de TWindow.
- Objet générique qui permet de représenter une icône dans une case, auquel on peut donner des propriétés et sur lequel on peut faire des actions.

♦ **Champs de l'objet :**

- Propriété : Personnalité (ensemble de qualités de l'objet « TIcône » telles que « pas_dessiné », « non_vu », « zéro_caché », ...).
- Icône : Table des icônes sur laquelle pointe l'objet pour pouvoir choisir le type d'icône que l'on veut représenter (normal_positif, normal_négatif, erreur, ...).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet TIcône.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X, Y	Integer	Coordonnées du coin sup. droit de l'icône
Propriétés	Personnalité	Qualités de l'icône
Icône	PTableIcône	Table d'icônes sur laquelle pointe l'objet

- **Procédure SetPropriété**

  Objet : Changement des propriétés de l'objet.

  Si S est vide alors ajouter SPlus et retirer SMoins à l'ensemble des Propriétés.

  Sinon, Propriétés = S

  Paramètres :

S, SPlus, SMoins	Personnalité	Ensembles de Propriétés
Etendue	Boolean	Permet d'étendre les Propriétés aux objets associés si Etendue = vrai

- **Fonction Sélectionné** : Boolean

  Objet : Rend vrai si l'état de l'icône est « sélectionné ».

- **Procédure Paint**

  Objet : Permet de dessiner l'icône à l'écran en fonction des propriétés.

  Paramètres :

PaintDC	HDC	Contexte d'affichage
PaintInfo	TPaintStruct	Structure indiquant les coordonnées de la région à redessiner

- **Procédure WMLButtonDown**

  Objet : Permet de réagir au click du bouton gauche de la souris sur l'icône.

  Paramètres :

Msg	TMessage	Paramètres du message WMLButtonDown

- **Procédure WMRButtonDown**

  Objet :Permet de réagir au click du bouton droit de la souris sur l'icône.

  Paramètres :

Msg	TMessage	Paramètres du message WMRButtonDown

- **Procédure WMMouseMove**

  Objet : Permet de réagir au déplacement de la souris (pour le drag and drop).

  Paramètres :

Msg	TMessage	Paramètres du message WMMouseMove

- **Procédure GetWindowClass**

  Objet : Donne les attributs de la classe de l'objet.

  Paramètres :

AWndClass	TWndClass	Attributs de la classe de l'objet

- **Fonction GetClassName**

  Objet : Donne le nom de la classe de l'objet.

3.2.1.2. Objet TIcôneRang :

♦ **Caractéristiques :**

- Objet enfant de TIcône.
- Objet qui permet de représenter une icône de type « Rang » (« bonbon », « sachet », « caisse », « armoire ») dans une case.

♦ **Champs de l'objet :**

- Rang : Interv_expo (rang de l'icône : 0 = unité, 1 = dizaine, 2 = centaine, 3 = millier).
- Associé : TListeRang (liste des objets « TIcôneRang » associée à l'objet).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X, Y	Integer	Coordonnées du coin sup. droit de l'icône
Rang	Interv_expo	Rang de l'icône
Propriétés	Personnalité	Qualités de l'icône

- **Destructor Eclate**

  Objet : Destructeur de l'objet Eclate l'objet en dix objets de rang directement inférieur.

  Paramètres :

Rct	TRect	Rectangle dans lequel on affiche les 10 nouveaux objets

- **Fonction Nombre** : Integer

  Objet : Compte le nombre d'icônes identiques qu'il y a dans la fenêtre qui contient l'objet.

- **Procédure WMLButtonDown**

  Objet : Sélection ou désélection de l'icône.

  Paramètres :

Msg	TMessage	Paramètres du message WMLButtonDown

3.2.1.3. Objet TListeRang :

♦ **Caractéristiques :**

- Objet enfant de TCollection.
- Objet reprenant une liste d'icônes de type « TIcôneRang » de même rang et leurs propriétés.

♦ **Champs de l'objet :**

- Cadre : TRect (rectangle dans lequel va se dessiner la liste).
- Fenêtre : TWindow (fenêtre contenant la liste).
- dx, dy : Shortint (écart entre les icônes).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

ALimit	Integer	Nombre maximum d'élément dans la liste
ADelta	Integer	Incrément possible si la liste devient trop petite

- **Procédure SetPropriété**

  Objet : Changement des propriétés d'une icône de la liste.

  Si S est vide alors ajouter SPlus et retirer SMoins à l'ensemble des Propriétés.

  Sinon, Propriétés = S

  Paramètres :

S, SPlus, SMoins	Personnalité	Ensembles de Propriétés
Etendue	Boolean	Permet d'étendre les Propriétés aux objets associés si Etendue = vrai

- **Fonction Tout_Sélectionné** : Boolean

  Objet : Rend vrai si tous les objets de la liste sont sélectionnés.

- **Fonction Somme** : Word

  Objet : Donne la somme des objets sélectionnés dans la liste.

- **Destructor Done**

  Objet : Détruit la liste.

3.2.1.4. Objet TIcôneChiffre:

♦ **Caractéristiques :**

- Objet enfant de TIcôneRang.
- Objet qui permet de représenter une icône de type chiffre (« 0 », « 1 » ..., « 9 ») dans une case.
- Chaque icône de type « chiffre » est d'un certain rang (0= unité, 1= dizaine, 2= centaine, 3= millier). C'est pourquoi l'objet TIcôneChiffre est un objet enfant de TIcôneRang.

♦ **Champs de l'objet :**

- Chiffre : Shortint (chiffre contenu dans l'icône).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X, Y	Integer	Coordonnées du coin sup. droit de l'icône
Rang	Interv_expo	Rang de l'icône
Propriétés	Personnalité	Qualités de l'icône
Valeur_num	Shortint	Valeur numérique du chiffre représenté par l'icône

- **Fonction Norme** : Shortint

  Objet : Normalisation d'un nombre supérieur ou égal à 10, donne le report comme résultat de la fonction.

- **Procédure SetValeur**

Objet : Donne une valeur numérique à une icône de type chiffre.

Paramètres :

Valeur_num	Shortint	Valeur numérique de l'icône

- **Procédure SetPropriété**

Objet : Changement des propriétés de l'objet.

Si S est vide alors ajouter SPlus et retirer SMoins à l'ensemble des Propriétés.

Sinon, Propriétés = S

Paramètres :

S, SPlus, SMoins	Personnalité	Ensembles de Propriétés
Etendue	Boolean	Permet d'étendre les Propriétés aux objets associés si Etendue = vrai

- **Procédure Paint**

Objet : Permet de dessiner l'icône à l'écran en fonction des propriétés.

Paramètres :

PaintDC	HDC	Contexte d'affichage
PaintInfo	TPaintStruct	Structure indiquant les coordonnées de la région à redessiner

- **Procédure WMLButtonDown**

Objet : Sélection et désélection de l'icône

Paramètres :

Msg	TMessage	Paramètres du message WMLButtonDown

- **Procédure WMChar**

Objet : Permet d'encoder un chiffre dans l'icône si il est correct.

Paramètres :

Msg	TMessage	Paramètres du message WMChar

- **Procédure Associe**

  Objet : Associe et dessine la représentation imagée de l'icône de type chiffre.

  Paramètres :

Fenêtre	PFenVisionDétail	Fenêtre de visualisation de l'opération détaillée
Rct	TRect	Rectangle de la fenêtre dans lequel on travail
EX, EY	Shortint	Ecart entre les icônes

3.2.1.5. Objet TBoutonSpécial:

♦ **Caractéristiques :**

- Objet enfant de TButton.
- Objet qui permet de représenter des boutons spéciaux avec un texte quelconque à l'intérieur.

♦ **Champs de l'objet :**

- Pressé : Boolean (vrai si le bouton a été pressé ou non).

♦ **Action de l'objet :**

- **Procédure WMLButtonDown**

  Objet : Permet de réagir au click du bouton gauche de la souris sur le bouton spécial (met le champ « pressé » à vrai quand le bouton spécial a été enfoncé).

  Paramètres :

Msg	TMessage	Paramètres du message WMLButtonDown

3.2.1.6. Objet TListeChiffre :

♦ **Caractéristiques :**

- Objet enfant de TListeRang.
- Objet reprenant une liste d'icônes « chiffres » de même rang et leurs propriétés.

♦ **Actions de l'objet :**

- **Fonction Tout_Sélectionné_Ou_Zéro** : Boolean

  Objet : Rend vrai si toutes les icônes de la liste sont sélectionnées ou s'il s'agit de zéros qui sont sélectionnés d'office.

- **Fonction Somme** : Word

  Objet : Donne la somme des chiffres contenus dans les icônes de la liste.

- **Fonction Un_Sélectionné** : Boolean.

  Objet : Rend vrai et un pointeur sur une icône « chiffre » sélectionnée dans la liste s'il y en a une.

  Paramètres :

P	PChiffre	Pointeur sur une icône chiffre

3.2.2. Les objets d'addition.

3.2.2.1. Objet TChiffreSomme :

♦ **Caractéristiques :**

- Objet enfant de TIcôneChiffre.
- Objet qui permet d'afficher et de calculer la somme de plusieurs chiffres (sous la forme calcul écrit et sous la forme imagée).

♦ **Champs de l'objet :**

- Etat_Avancement : Byte (état d'avancement de l'opération d'addition de chiffres).
- Liste : PListeChiffre (liste des chiffres à additionner).
- Report : PChiffre (report de l'addition des chiffres).
- Emballe : PBoutonSpécial (bouton qui permet d'emballer 10 icônes de même rang).
- Embx, Emby : Integer (coordonnées du bouton « Emballe »).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X, Y	Integer	Coordonnées du coin sup. droit de l'icône
Rang	Interv_expo	Rang de l'icône
Propriétés	Personnalité	Qualités de l'icône
Valeur_num	Shortint	Valeur numérique du chiffre représentée par l'icône
Liste	PListeChiffre	Pointeur sur la liste des chiffres à additionner

Report	PChiffre	Pointeur sur le report de l'addition des chiffres
EmbX, EmbY	Integer	Coordonnées du coin sup. droit du bouton « Emballe »

- **Procédure Débute_Opération**

  Objet : Initialisation de l'opération d'addition des chiffres.

- **Procédure WMTimer**

  Objet : Effectue l'opération d'addition des chiffres.

  Paramètres :

Msg	TMessage	Paramètres du message WMTimer

- **Fonction Opération_Finie** : Boolean

  Objet : Rend vrai si l'opération d'addition est finie.

- **Procédure IdEmballe**

  Objet : Remplace dix objets de même rang par un objet de rang directement supérieur

  Paramètres :

Msg	TMessage	Paramètres du message WMLButtonDown

3.2.2.2. Objet TNombre :

♦ **Caractéristiques :**

- Objet enfant de TWindow.
- Objet qui permet de représenter un nombre sous forme chiffrée.

♦ **Champs de l'objet :**

- Nombre : tableau de PChiffre.

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X, Y	Integer	Coordonnées du coin sup. droit de l'icône
NS	String	Valeur du nombre sous forme de string
Propriétés	Personnalité	Qualités de l'icône

- **Procédure Met_à_Zéro**

  Objet : Met tous les chiffres d'un nombre zéro.

- **Procédure Normalise**

  Objet : Normalise un nombre.

- **Procédure Associe**

  Objet : Associe une représentation imagée au nombre.

  Paramètres :

Fenêtre	PFenVisionDétail	Fenêtre de visualisation de l'opération détaillée
Rect	TRect	Rectangle de la fenêtre dans lequel on travaille

- **Destructor Done**

  Objet : Destruction de l'objet.

3.2.2.3. Objet TListeNombre :

♦ **Caractéristiques :**

- Objet enfant de TCollection.
- Objet reprenant une liste de nombres.

♦ **Actions de l'objet :**

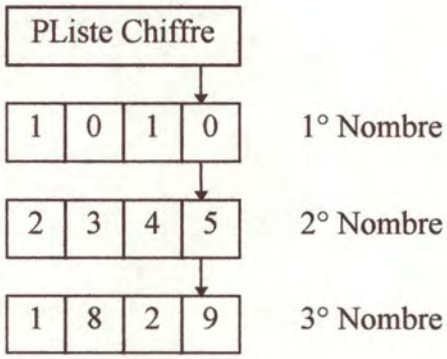
- **Fonction ListeChiffre** : PListeChiffre

  Objet : Donne la liste des chiffres d'un rang donné des nombres contenus dans TListeNombre.

  Paramètres :

Rang	Interv_expo	Rang des chiffres à choisir

Ex :



La liste des chiffres de rang 0 est : 0 , 5 , 9..

3.2.2.4. Objet TNombreSomme :

♦ **Caractéristiques :**

- Objet enfant de TNombre.
- Objet qui permet de faire la somme de plusieurs nombres.

♦ **Champs de l'objet :**

- Etat_Avancement : Byte (état d'avancement de l'opération).
- Liste : PListeNombre (liste des nombres à additionner).
- Report : PNombre (pointeur sur un TNombre qui reprend tous les reports de l'addition).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X, Y	Integer	Coordonnées du coin sup. droit de l'icône
Propriétés	Personnalité	Qualités de l'icône
Liste	PListeNombre	Pointeur sur la liste des nombres à additionner
Report	PNombre	Pointeur sur le nombre qui reprend les reports de l'addition des nombres
EmbX, EmbY	Integer	Coordonnées du coin sup. droit du bouton « Emballe »

- **Procédure Débute_Opération**

  Objet : Initialisation de l'opération d'addition des nombres.

- **Procédure Opération_Finie** : Boolean

  Objet : Rend vrai si l'opération d'addition est finie ou non.

- **Procédure WMTimer**

  Objet : Effectue l'opération d'addition des nombres.

  Paramètres :

Msg	TMessage	Paramètres du message WMTimer

3.2.2.5. Objet TFenêtreVisionDétail :

♦ **Caractéristiques :**

- Objet enfant de TWindow.
- Objet permettant de visionner le déroulement de l'opération d'addition sous forme imagée.

♦ **Champs de l'objet :**

- ListeRect : PCollection (rectangles dans lesquels on dessine les nombres sous forme imagée).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
ATitle	PChar	Titre de la fenêtre de visualisation du calcul détaillé
Emb	Boolean	Indique si le bouton « Emballe » a été enfoncé
XEmb, YEmb	Integer	Coordonnées du coin sup. droit du bouton « Emballe »

- **Procédure Insère**

  Objet : Insérer un objet « TRect » dans la liste ListeRect.

  Paramètres :

TRect	PRectObj	Rectangle qui va se dessiner

- **Procédure Paint**

  Objet : Dessine la fenêtre et son contenu.

  Paramètres :

PaintDC	HDC	Contexte d'affichage
PaintInfo	TPaintStruct	Structure indiquant les coordonnées de la région à redessiner

- **Destructor Done**

  Objet : Destruction du bouton Emballe et de la liste ListeRect.

3.2.3. Les objets de multiplication.

3.2.3.1. Objet TBoîte :

♦ **Caractéristiques :**

- Objet enfant de TRang.
- Objet permettant de représenter une icône de type « boîte » et d'ouvrir cette boîte.

♦ **Champs de l'objet :**

- Nba : PNombre (pointeur vers un TNombre associé à la boîte qui représente le multiplicande).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X , Y	Integer	Coordonnées du coin sup. droit de l'icône
Rang	Interv_expo	Rang de l'icône
Propriété	Personnalité	Qualités de l'icône
Nbassocié	PNombre	Pointeur vers un TNombre associé à la boîte

- **Procédure HMinRect** : Byte

  Objet : Donne la hauteur minimale du rectangle nécessaire pour afficher le contenu de la boîte.

- **Procédure WMLButtonDown**

  Objet : Ouverture de la boîte grâce à la procédure Associe du TNombre associé à la boîte.

  Paramètres :

Msg	TMessage	Paramètres du message WMLButtonDown

3.2.3.2. Objet TNombreFictif :

♦ **Caractéristiques :**

- Objet enfant de TNombre.
- Objet permettant de représenter le multiplicande (éventuellement multiplier par un multiple de dix) sous forme imagée.

♦ **Champs de l'objet :**

- Rang : Interv_expo (rang du chiffre du multiplicateur considéré).

♦ **Actions de l'objet :**

- **Procédure Duplique**

  Objet : Crée un nouvel objet et copie le multiplicande dedans.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X , Y	Integer	Coordonnées du coin sup. droit de l'icône
Rang	Interv_expo	Rang de l'icône
Propriété	Personnalité	Qualités de l'icône
Egal	PNombre	Pointeur vers un TNombre associé à la boîte

- **Procédure Associe**

  Objet : Associe à la boîte la représentation imagée du multiplicande.

  Paramètres :

Fenêtre	PFenVisionDetail	Fenêtre de visualisation du calcul détaillé
Rct	TRect	Rectangle dans lequel se dessine la représentation imagée du multiplicateur

3.2.3.3. Objet TNombreFois :

♦ **Caractéristiques :**

- Objet enfant de TNombreSomme.
- Objet permettant de multiplier un chiffre par un nombre et de visualiser le calcul détaillé de cette opération.
- Le produit d'un nombre par un chiffre est vu comme une somme de plusieurs nombres, tous égaux au multiplicande.

♦ **Champs de l'objet :**

- Multiplicande : PNombre (pointeur vers une liste qui représente le multiplicande).
- ChiffreMultiplicateur : PChiffre (pointeur vers un Pchiffre qui représente le multiplicateur).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X , Y	Integer	Coordonnées du coin sup. droit de la fenêtre
Propriété	Personnalité	Qualités de l'objet
M_ateur	PChiffre	Pointeur vers un PChiffre qui contient le multiplicateur
M_ande	PNombre	Pointeur vers un PNombre qui contient le multiplicande
Report	PNombre	Pointeur vers un PNombre qui contient la liste de tous les reports
XEmb, YEmb	Integer	Coordonnées du coin sup. droit du bouton « Emballe »

- **Procédure WMTimer**

  Objet : Effectue l'opération de multiplication d'un nombre par un chiffre.

  Paramètres :

Msg	TMessage	Paramètres du message WMTimer

3.2.3.4. Objet TNombreProd :

♦ **Caractéristiques :**

- Objet enfant de TNombreSomme.
- Objet permettant de multiplier un nombre par un nombre et de visualiser le calcul détaillé de cette opération.
- Le produit d'un nombre par un nombre est vu comme une somme de plusieurs mulitplications.

♦ **Champs de l'objet :**

- Multiplicande : PNombre (pointeur vers un PNombre qui représente le multiplicande).
- Multiplicateur : PNombre (pointeur vers un PNombre qui représente le multiplicateur).

♦ **Actions de l'objet :**

- **Constructor Init**

  Objet : Initialisation de l'objet.

  Paramètres :

AParent	PWindowsObject	Fenêtre parent de l'objet
X , Y	Integer	Coordonnées du coin sup. droit de la fenêtre
Propriété	Personnalité	Qualités de l'objet
M_ateur	PNombre	Pointeur vers un PNombre qui contient le multiplicateur
M_ande	PNombre	Pointeur vers un PNombre qui contient le multiplicande
Report	PNombre	Pointeur vers un PNombre qui contient la liste de tous les reports
XEmb, YEmb	Integer	Coordonnées du coin sup. droit du bouton « Emballe »

- **Procédure WMTimer**

  Objet : Effectue l'opération de multiplication d'un nombre par un nombre.

  Paramètres :

Msg	TMessage	Paramètres du message WMTimer

3.3. Le gestionnaire d'événements.

3.3.1. Description du gestionnaire d'événements.

Comme on l'a vu dans l'analyse pédagogique, lors du déroulement d'une opération, c'est-à-dire, lors du déroulement d'un programme, de nombreux événements se produisent. Lorsqu'un événement se produit (par exemple, lorsqu'un élève tente d'ouvrir un objet « bonbon » à l'aide de la souris), le gestionnaire doit y répondre (par exemple, en signalant à l'élève qu'il ne peut pas ouvrir un « bonbon »). Le gestionnaire doit tenir compte de l'événement qui s'est produit, de l'état d'avancement de l'opération et du niveau des connaissances de l'élève. La réponse du gestionnaire est l'envoi d'un message à l'élève qui a engendré l'événement.

Pour gérer les événements en fonction de l'état d'avancement de l'opération en cours, le gestionnaire doit avoir accès à un certain nombre de renseignements. Ces renseignements sont repris dans un tableau que le gestionnaire consultera pour décider de quelle manière il doit répondre à l'événement. J'ai analysé deux représentations informatiques possibles de ce tableau. Elles sont détaillées ci-dessous :

- Le tableau à deux entrées :

  La première entrée du tableau reprend tous les événements possibles, la seconde reprend tous les états possibles du programme.

Etat / Evén.	Et 1	Et 2
Ev 1	Act 1	Act 2
Ev 2	Act 3	Act 1

Pour trouver l'action à effectuer, il suffit de regarder à l'intersection de l'événement et de l'état voulus.

- Le tableau séquentiel :

Chaque ligne de ce tableau séquentiel reprend un couple événement-état suivi de l'action à accomplir.

Evénement	Etat	Action

Pour trouver l'action qu'il faut réaliser, il suffit de passer tout le tableau en revue jusqu'à ce qu'on trouve le bon couple événement-état.

J'ai choisi d'utiliser la deuxième représentation car, lorsqu'il y a des couples événement-état qui ne donnent aucune action à accomplir, on peut purement et simplement supprimer l'entrée correspondante du tableau, ce qui n'est pas le cas pour le tableau à deux entrées.

La représentation choisie, de même que le tableau à deux entrées, ne permet pas de tenir compte du niveau de l'élève. C'est pourquoi dans l'analyse pédagogique, nous avons introduit la notion de fréquence de production des événements. Lorsqu'un événement se produit, il est comptabilisé dans le tableau séquentiel et le gestionnaire ne répondra à cet événement qu'au moment où ce nombre atteindra la limite (voir analyse pédagogique, point 2.3.5.4.). La limite est donc aussi stockée dans le tableau.

Les actions à accomplir sont l'envoi de messages. Pour déterminer le message à envoyer, le tableau doit aussi donner au gestionnaire le numéro du message à envoyer. Tous ces renseignements sont repris dans le tableau séquentiel (TTableMessage) à la suite de chaque couple événement-état comme décrit ci-dessous :

N° Evénement	Etat	Type message	Compteur	Limite	N° message

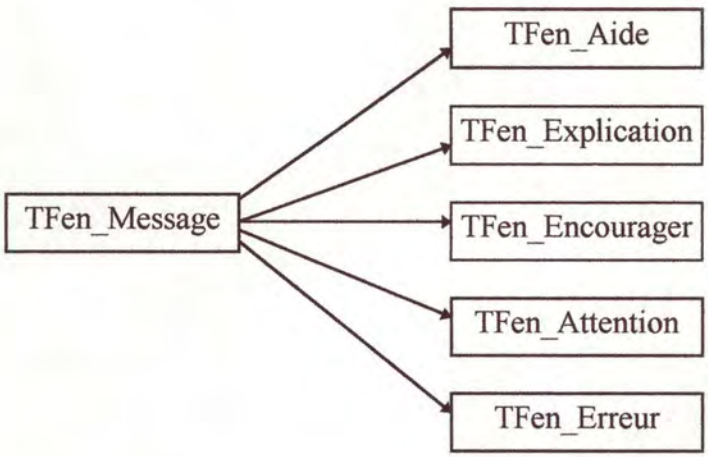
- Le numéro d'événement : il détermine un et un seul événement.
- L'état d'avancement : il détermine l'état dans lequel se trouve le programme. Combiné au numéro d'événement, il permet d'accéder à une ligne du tableau séquentiel.
- Le type de message : il détermine si le message à afficher, lorsque le couple événement-état se produit, est un message d'aide, de mise en garde ou d'erreur. Il est choisi par le professeur.
- Le compteur : il permet de comptabiliser le nombre de fois qu'un même couple événement-état s'est produit. Quand le compteur atteint la limite d'affichage, le message est affiché et le compteur est remis à zéro.
- La limite d'affichage : elle est choisie par le professeur en fonction du niveau de l'élève qui utilise le didacticiel.
- Le numéro de message : à chaque couple événement-état correspondra un numéro de message. Ce numéro permettra à la fenêtre d'affichage d'accéder au bon message à afficher.

Pour illustrer le contenu du tableau TTableMessage, voici un exemple de ce qui pourrait se passer pendant l'exécution du programme : supposons que l'élève a terminé sa multiplication (état du programme) mais qu'il clique toutefois dans une des fenêtres à icônes (événement). Si le professeur qui rempli le tableau TTableMessage décide qu'après trois tentatives de ce genre, le gestionnaire doit envoyer le message n°1 d'aide à l'utilisation du didacticiel suivant : « La multiplication est terminée. Pour faire un nouvel exercice, tu dois enfoncer le bouton " Nouvelles valeurs " », alors il remplira la ligne qui correspond au couple événement-état comme suit :

Evénement 1	Etat 1	Aide	0	3	Message 1

Après trois tentatives de l'élève, le compteur courant de cette ligne du tableau sera égal à 3 et le gestionnaire ordonnera l'affichage du message n°1.

Puisque l'affichage diffère suivant le type de message, j'ai créé cinq objets qui reprennent les différentes caractéristiques des trois types d'affichage. Néanmoins, ces cinq objets ont des points communs : la recherche du message à afficher dans le fichier de message, la mise en place du message dans la fenêtre d'affichage et la réception du n° de message à afficher. J'ai donc reprit ces propriétés communes dans un objet parent des cinq autres.



Ce procédé permettra de rajouter facilement un nouveau type de message en créant un objet enfant de TFen_Message et proche des cinq objets d'affichage.

3.3.2. Fonctionnement du gestionnaire d'événements.

Le gestionnaire d'événements sera capable de recevoir des messages WM_Signal de tous les objets constituants le programme (TIcôneRang, TIcôneChiffre, TNombre,...). Il aura accès au tableau TTableMessage, décrit ci-dessus, et pourra décider s'il faut ou non répondre à l'événement. Il pourra également envoyer des messages WM_Afficher aux cinq objets d'affichage.

Lorsque le gestionnaire recevra un WM_Signal, ayant comme paramètres le numéro de l'événement qui s'est produit et l'état dans lequel se trouve le programme, il se réferera dans le tableau TTableMessage. S'il trouve le bon couple événement-état, il incrémentera le compteur de cette ligne de 1. Il regardera ensuite la limite. Si celle-ci est égale à -1, il ne fera rien. Puis il comparera le compteur à la limite. Si le compteur est plus petit que la limite, il ne fera rien ; si le compteur est égal à la limite, il enverra un WM_Afficher à un des objets d'affichage. Il choisira celui-ci en consultant le type du message à afficher. Le message WM_Afficher aura comme paramètre le numéro du message à afficher. L'objet d'affichage qui recevra ce message pourra ensuite aller chercher le bon message dans son fichier de message et pourra l'afficher. L'envoi des messages WM_Signal et WM_Afficher s'effectuera grâce à la fonction PostMessage.

3.3.3. Description des objets de gestion et d'affichage.

3.3.3.1. Objet TFenêtreMessage :

♦ **Caractéristiques :**

- Fenêtre parent de TFen_Aide, TFen_Attention et TFen_Erreur.
- Fenêtre enfant de TWindow.

♦ **Actions de l'objet :**

- Accéder au fichier « messages ».
- Rechercher le message à afficher.
- Changer le message dans la fenêtre.
- Placer le message dans la fenêtre.
- Recevoir un message WM_Afficher du gestionnaire.

3.3.3.2. Objet TFenêtreAide et TFenêtreExplication :

♦ **Caractéristiques :**

- Fenêtre enfant de TFen_Message.
- Permettent d'afficher respectivement des messages d'aide ou d'explication sur le déroulement de l'opération dans une fenêtre et de changer ces messages quand c'est nécessaire.

♦ **Action de l'objet :**

- Initialiser la fenêtre.

3.3.3.3. Objets TFenêtreEncourager, TFenêtreAttention et TFenêtreErreur :

♦ **Caractéristiques :**

- Fenêtres enfants de TFen_Message.
- Permettent d'afficher respectivement des messages d'encouragement lors de bonnes réponses, de mise en garde et d'erreur dans une fenêtre et de faire apparaître cette fenêtre quand c'est nécessaire.

♦ **Actions des objets :**

- Initialise la fenêtre.
- Faire apparaître la fenêtre.
- Faire disparaître la fenêtre (acquitter le message, fermer la fenêtre).

3.3.3.4. Objet TGestionnaire :

♦ **Caractéristiques :**

- Fenêtre enfant de TWindow.

♦ **Actions de l'objet :**

- Incrémenter le compteur courant.
- Comparer la limite d'affichage et le compteur courant (décider s'il faut répondre ou non à l'événement).
- Envoyer un WM_Afficher (Paramètre : N° Message) à la fenêtre d'affichage adéquate.
- Recevoir un WM_Signal des objets dans lesquels se produisent les événements.

3.4. Les améliorations possibles à court terme.

3.4.1. Bouton « normaliser ».

Nous avons expliqué plus haut l'utilité du bouton « emballer ». Celui-ci permet de remplacer 10 objets de même rang, sélectionnés par l'élève, par un objet de rang supérieur. Ce bouton est très intéressant pour les élèves qui en connaissent le principe. Pour les autres, il ne sert à rien. Si on veut aller plus loin, on peut créer un bouton « normaliser ». Il permettrait de regrouper au maximum les objets de rang courant.

Ce bouton est très utile lorsqu'il y a plusieurs regroupements à faire. Par exemple, s'il y a 32 « bonbons » à regrouper, il faut répéter trois fois la même opération d'emballage de 10 bonbons pour obtenir les trois sachets-reports si l'on utilise le bouton « emballer ». Par contre, avec le bouton « normaliser », il suffit d'une seule opération. Une simple pression de ce bouton et les 30 bonbons sont remplacés par 3 sachets.

3.4.2. Le gestionnaire d'événements.

On peut améliorer le gestionnaire d'événements en lui permettant de tenir compte du niveau des connaissances des élèves. Il faudrait tenir compte de l'évolution des élèves puisque leur niveau de connaissances évolue au cours des exercices. Pour cela, il faudrait enregistrer toutes les actions et les erreurs des élèves. Ces enregistrements constitueraient un dossier sur lequel le gestionnaire se baserait pour affiner ses décisions.

Ainsi, le tableau séquentiel TTableMessage aurait comme entrée un triplet et non plus un couple. Ce triplet serait constitué de l'événement, de l'état et du niveau des connaissances de l'élève et c'est ce triplet qui déterminerait l'action à accomplir.

Evénement	Etat	Niveau des connaissances	Action à accomplir

Le niveau des connaissances des élèves est important. En effet, il n'est pas conseillé de répondre de la même façon à deux élèves de niveaux différents. Si l'on prend comme exemple l'événement « click souris dans une fenêtre » et l'état « l'opération est terminée », et supposons que l'un des deux élève sait que la multiplication est terminée mais qu'il ne sait pas comment il doit faire pour recommencer une nouvelle multiplication et que l'autre ne sait pas qu'elle est terminée. Avec le système du compteur et de la limite, le gestionnaire répondait à ce couple « événement-état » par la même action (par exemple, l'envoi du message « la multiplication est terminée »). Mais si l'on tient compte du niveau des connaissances des deux élèves, on peut leur répondre par des messages différents et donc mieux appropriés à leur problème.

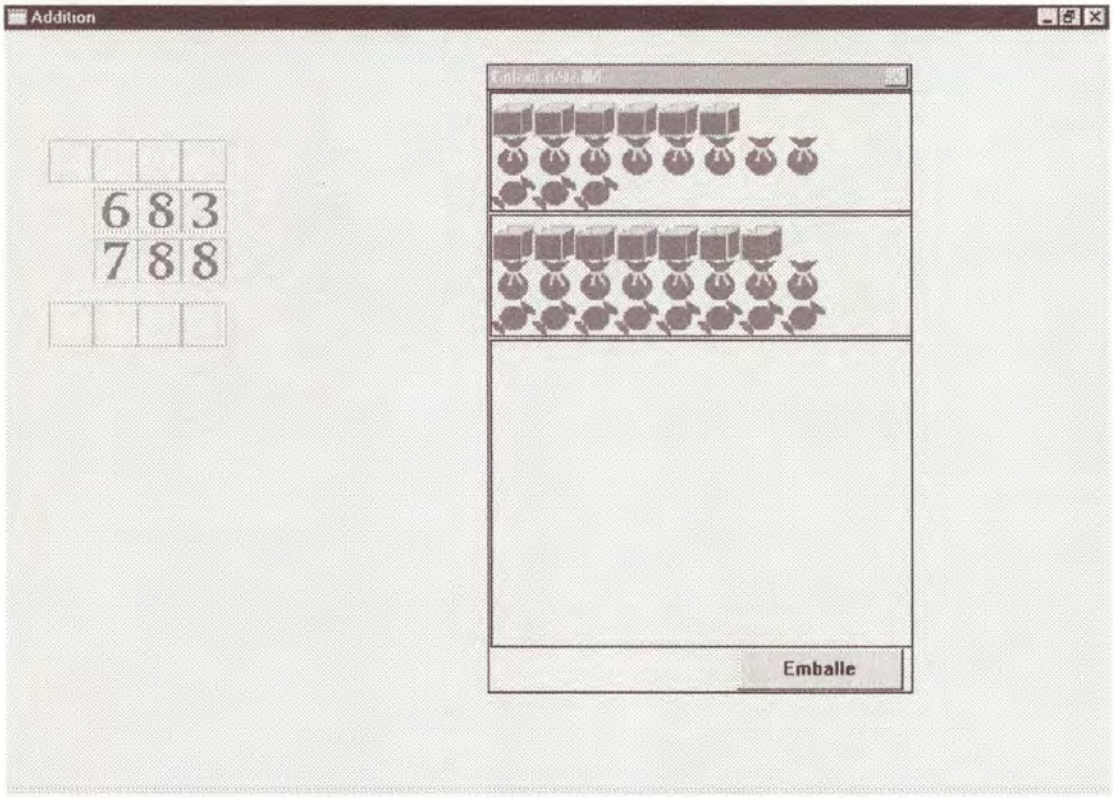
4

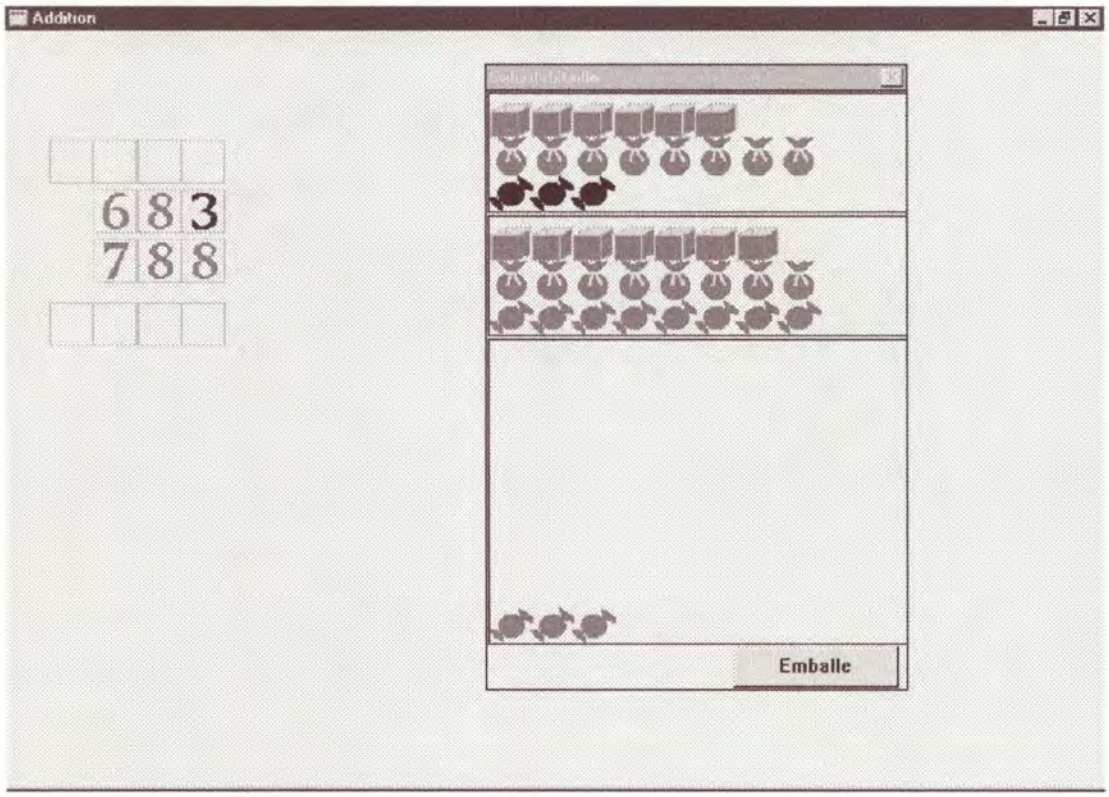
UTILISATION DES OBJETS DANS UN PROGRAMME

Un exemple d'addition.

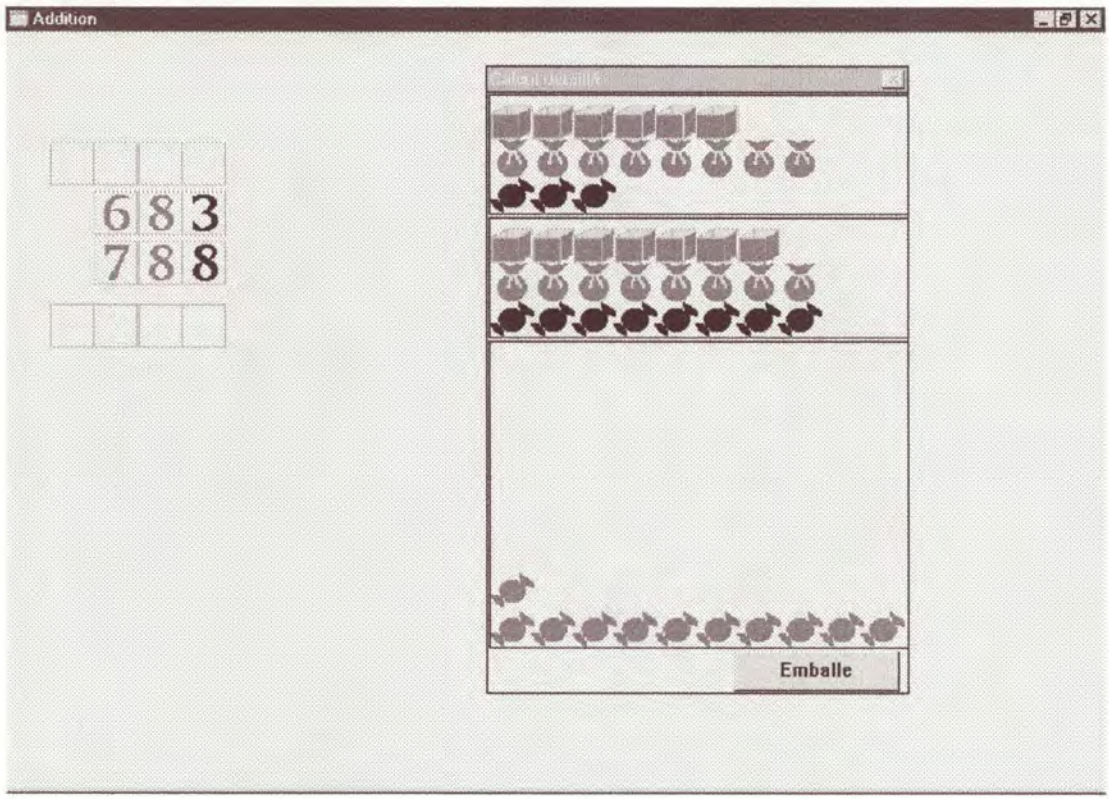
Pour montrer l'utilisation possible des objets dans un programme, voici un exemple du déroulement d'une opération d'addition. Celle-ci peut se faire sous forme imagée (c'est-à-dire avec les bonbons) ou sous forme de calcul écrit. Dans cet exemple, l'addition peut se faire des deux manières en même temps. Les élèves qui auraient des difficultés à comprendre le calcul écrit de cette addition pourraient passer sans difficultés à la représentation imagée.

Sur l'écran ci-dessous, la fenêtre « calcul détaillé » contient les deux nombres à additionner. L'élève doit sélectionner les bonbons du premier nombre et les bonbons du second nombre pour pouvoir les regrouper et obtenir le résultat au rang des unités et le report du rang des dizaines. Pour sélectionner ces bonbons, l'élève peut soit cliquer sur les chiffres correspondants du calcul écrit, soit cliquer sur les bonbons dans le calcul détaillé.

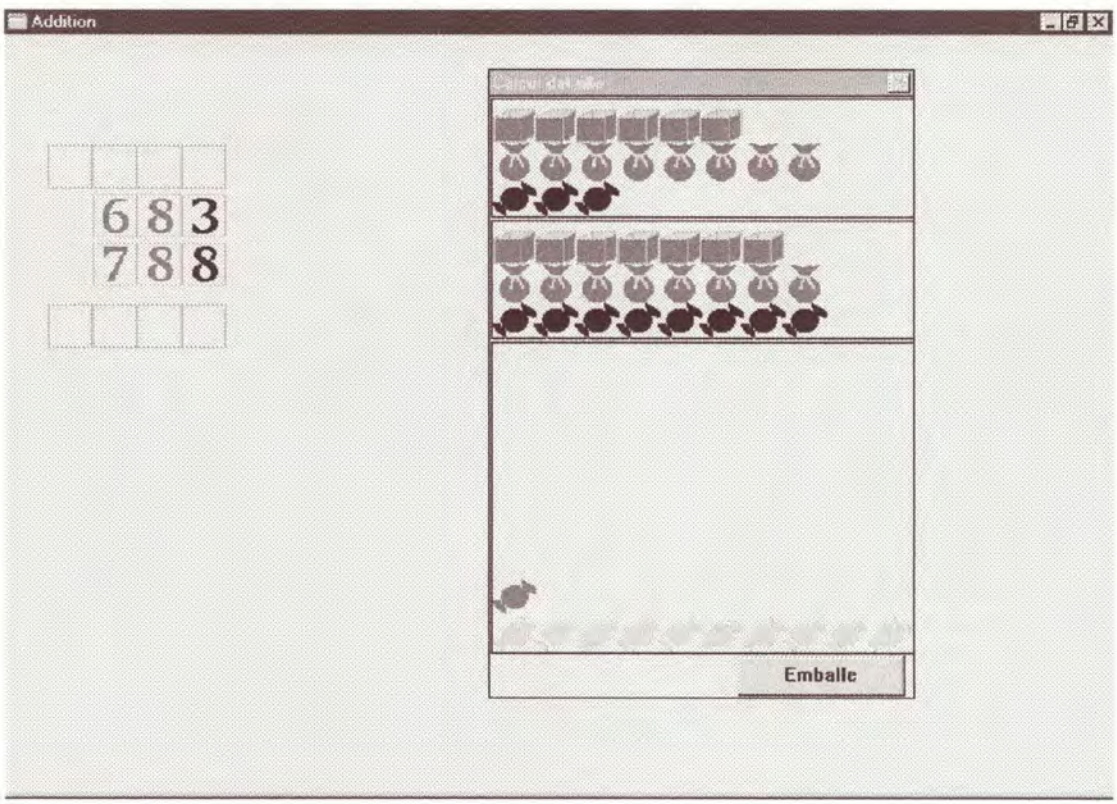




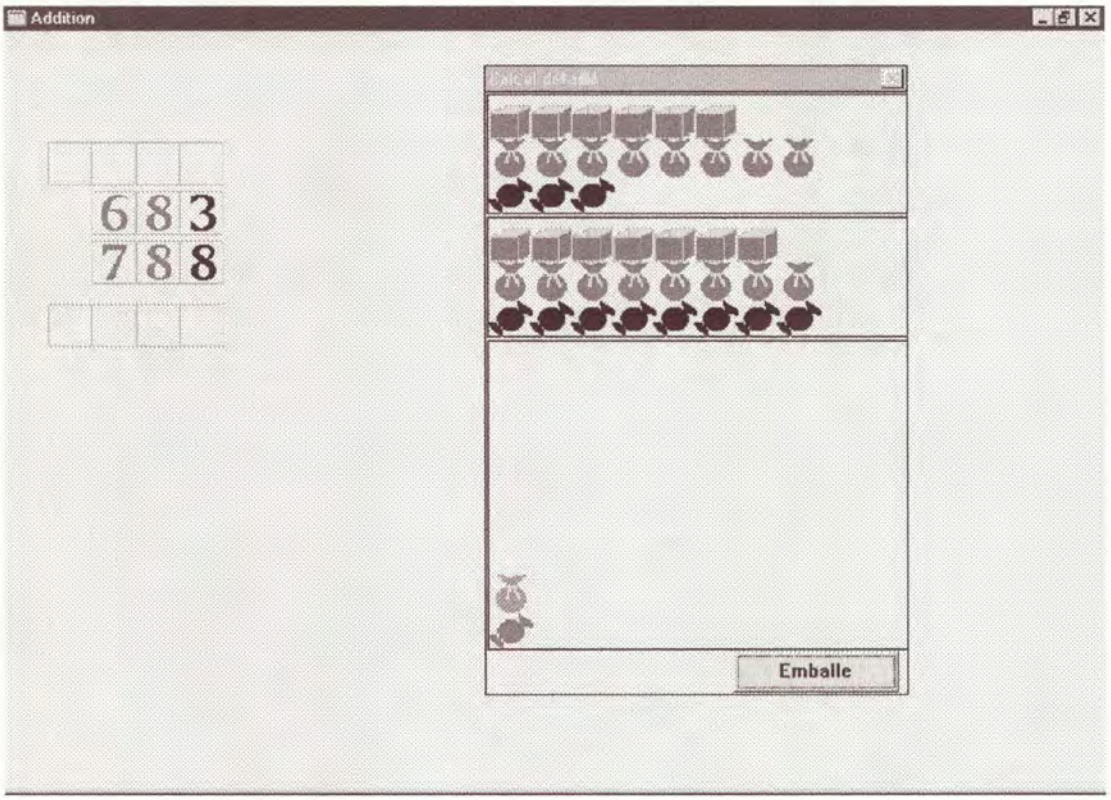
Les bonbons du premier nombre sont sélectionnés. Il faut sélectionner les bonbons du second nombre.



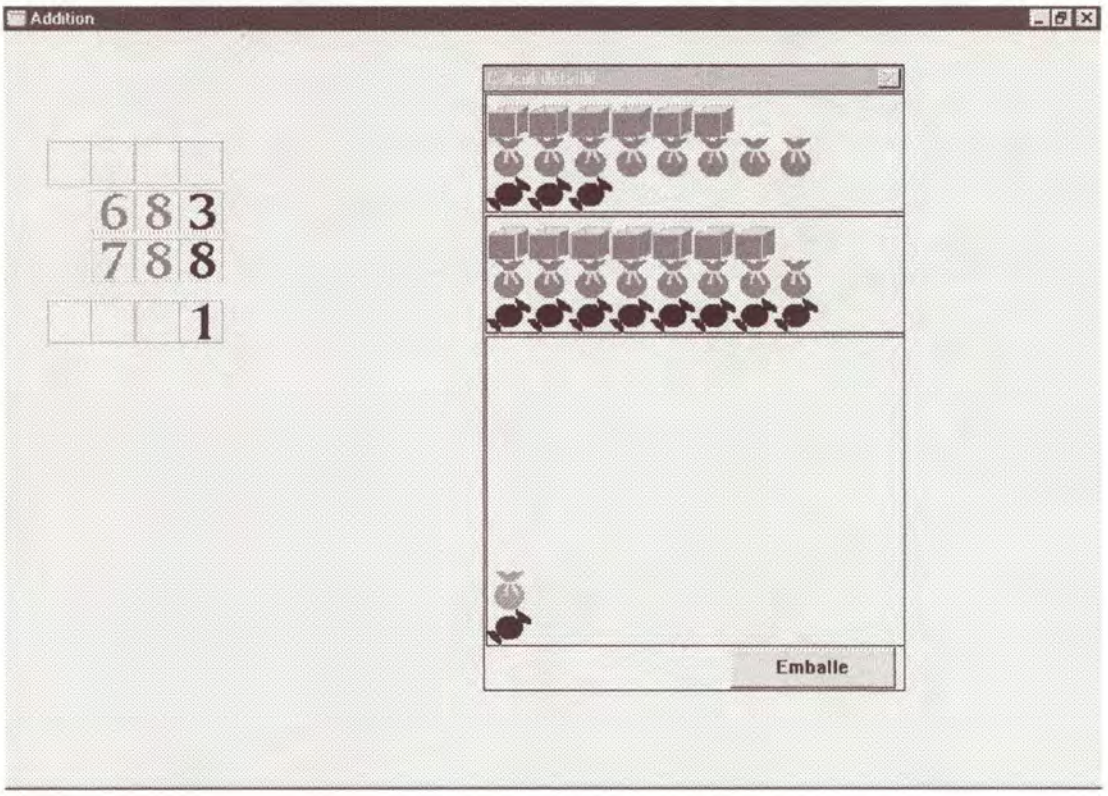
Il y a plus de 10 bonbons au total, donc l'élève ne peut pas écrire le résultat au rang des unités directement. Il doit d'abord procéder au regroupement de 10 bonbons pour obtenir un sachet report. Pour cela, il sélectionne 10 bonbons.



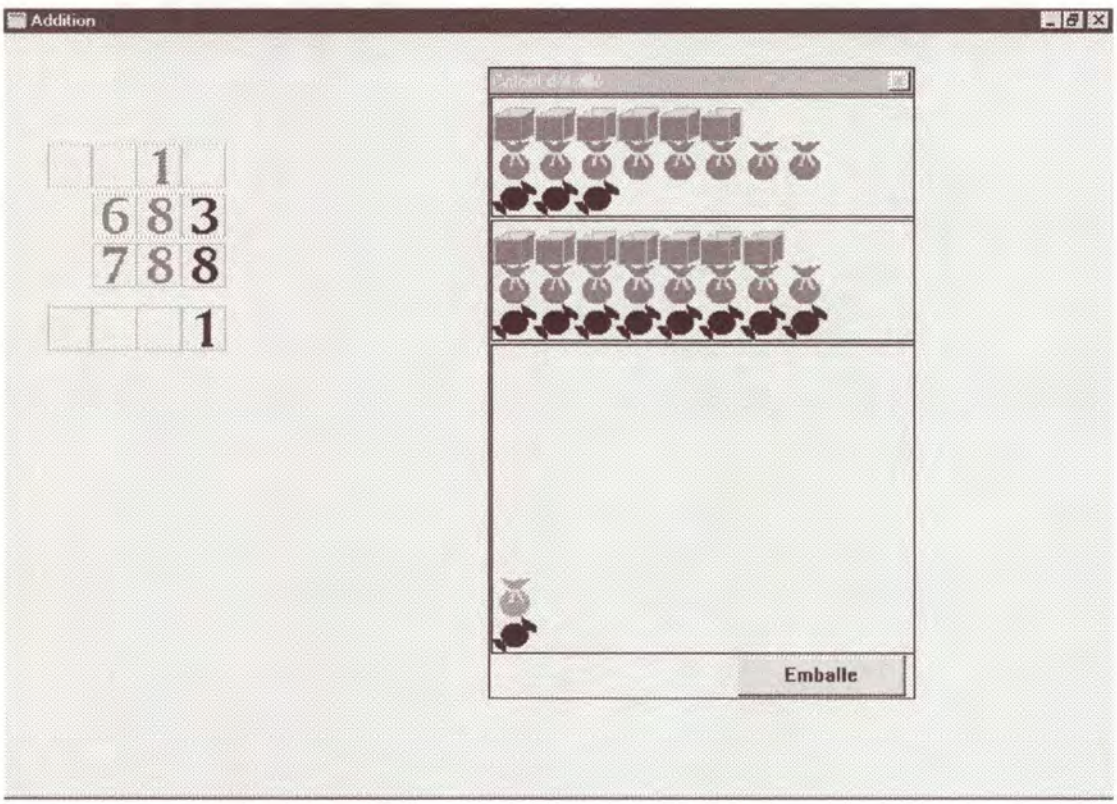
L'élève peut maintenant enfoncer le bouton « emballe » pour emballer les bonbons sélectionnés dans un sachet. Il lui reste donc un bonbon.



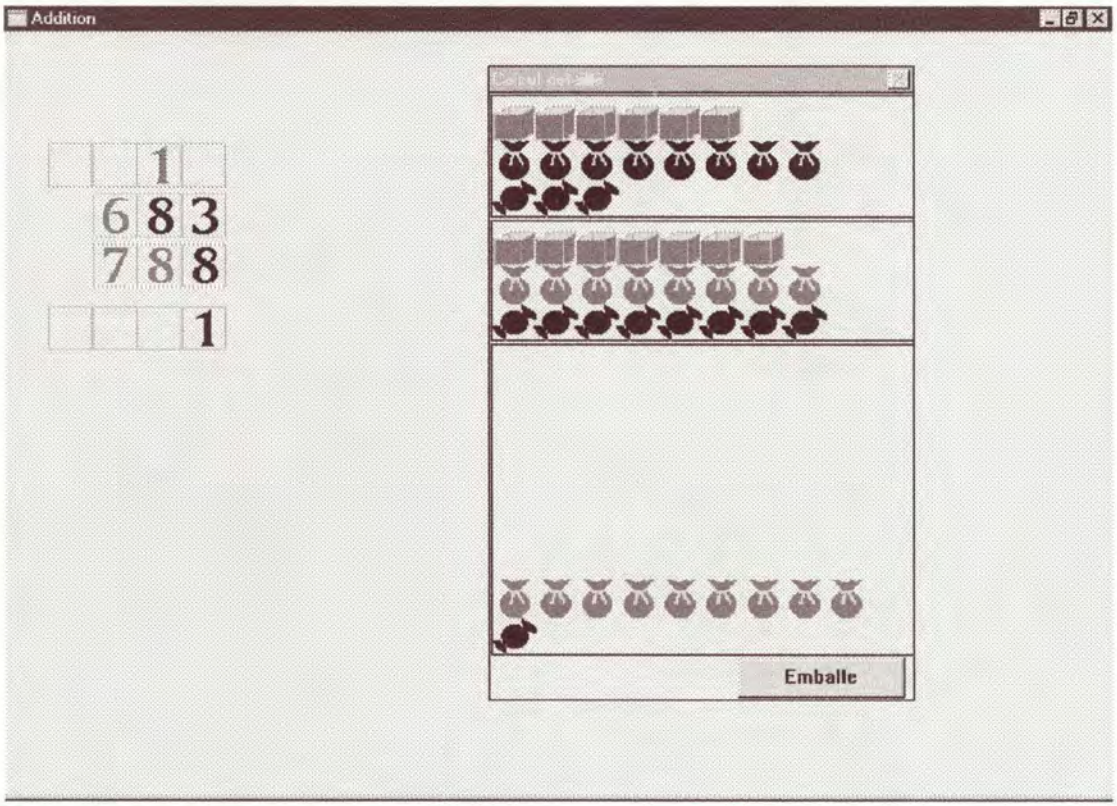
L'élève peut sélectionner le bonbon pour faire apparaître le résultat au rang des unités.



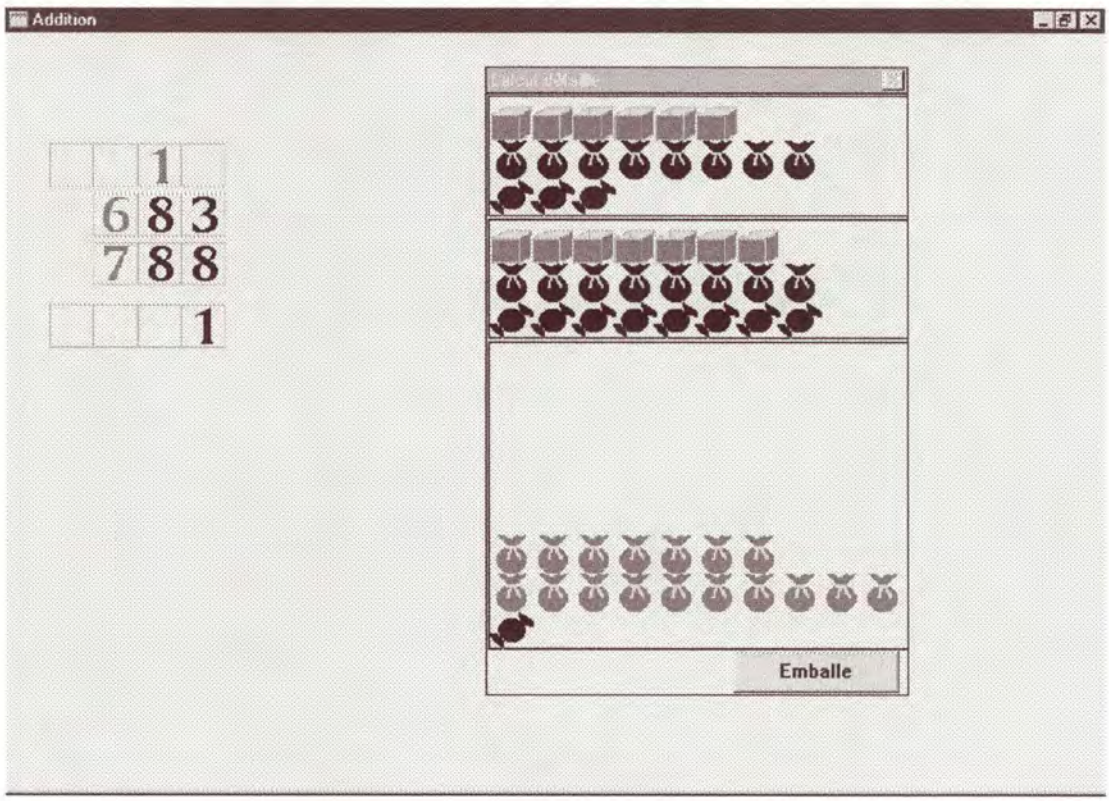
Il peut maintenant écrire le report du rang des dizaines dans le calcul écrit soit en sélectionnant le sachet-report à l'aide de la souris, soit en sélectionnant la case de report voulue et en écrivant le résultat dedans. L'élève a toujours cette possibilité de choisir le mode de sélection qu'il préfère (dans le calcul détaillé ou dans le calcul écrit). S'il choisit de faire le calcul sous forme imagée, le calcul écrit se remplit automatiquement en fonction des actions que l'élève fait dans le calcul détaillé et inversement.



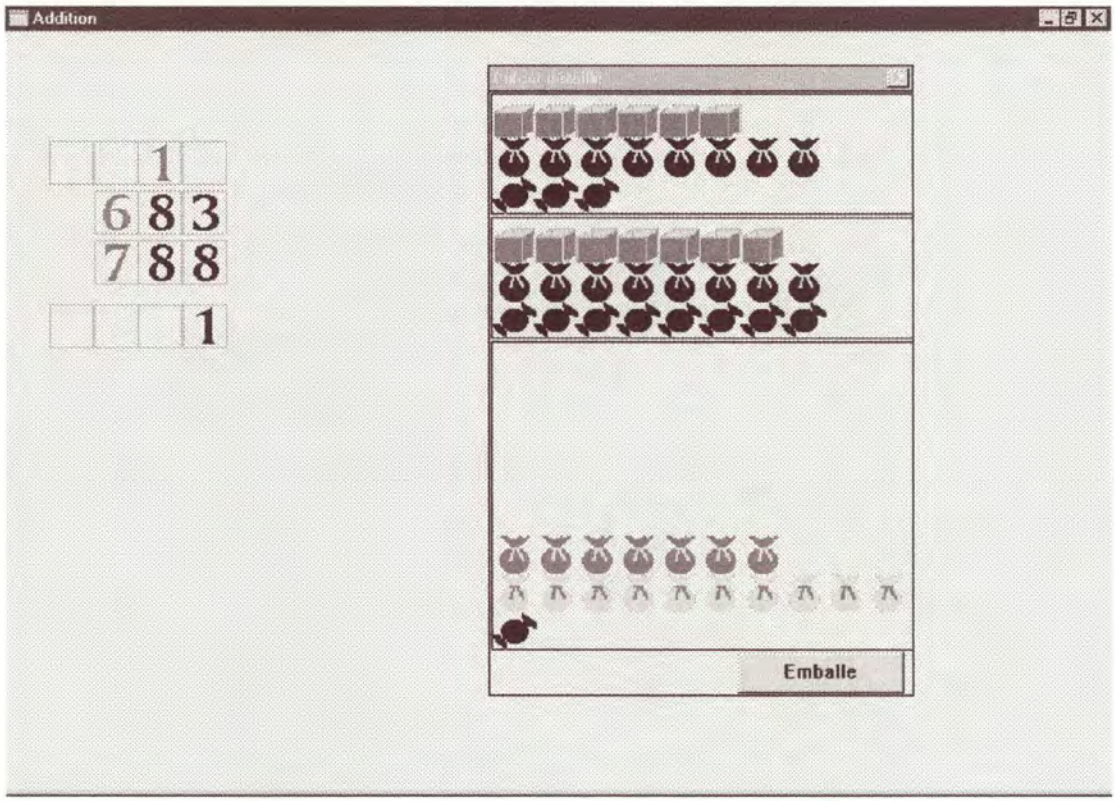
Le rang courant est maintenant la dizaine. L'élève doit sélectionner tous les chiffres du rang des dizaines qu'il doit additionner. Il sélectionne donc le chiffre des dizaines des deux nombres et le report.



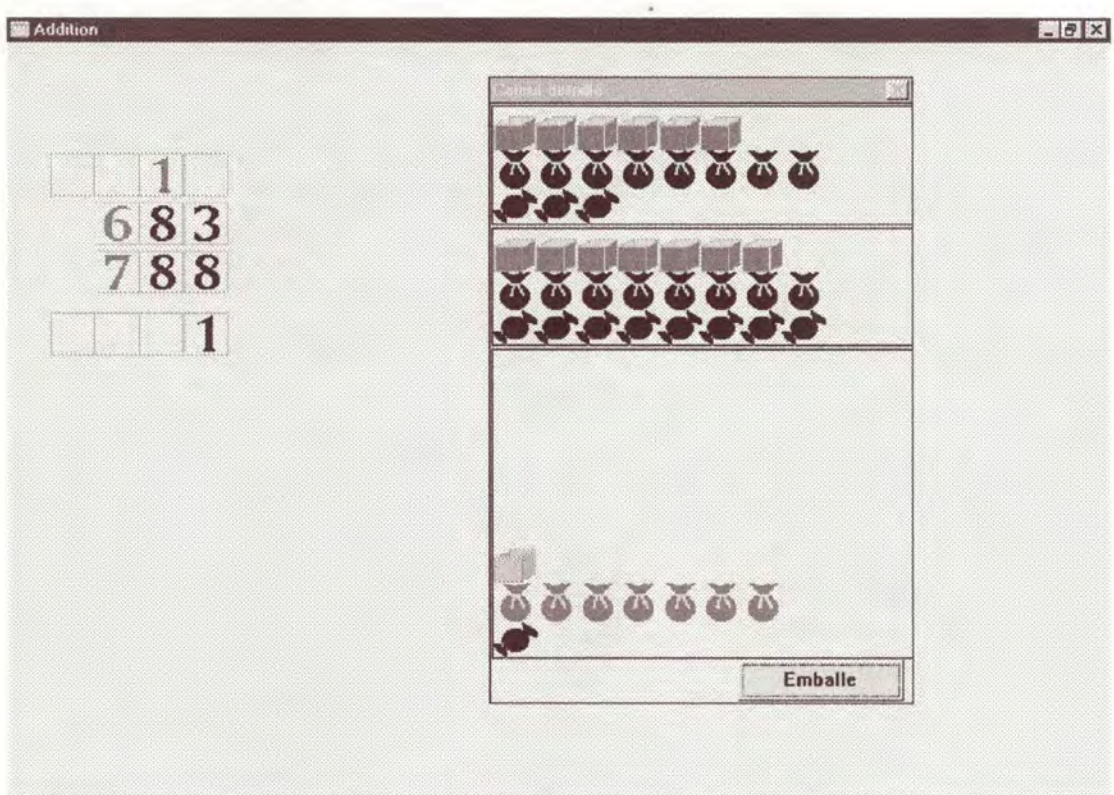
Le sachet-report et les sachets du premier nombre sont sélectionnés.



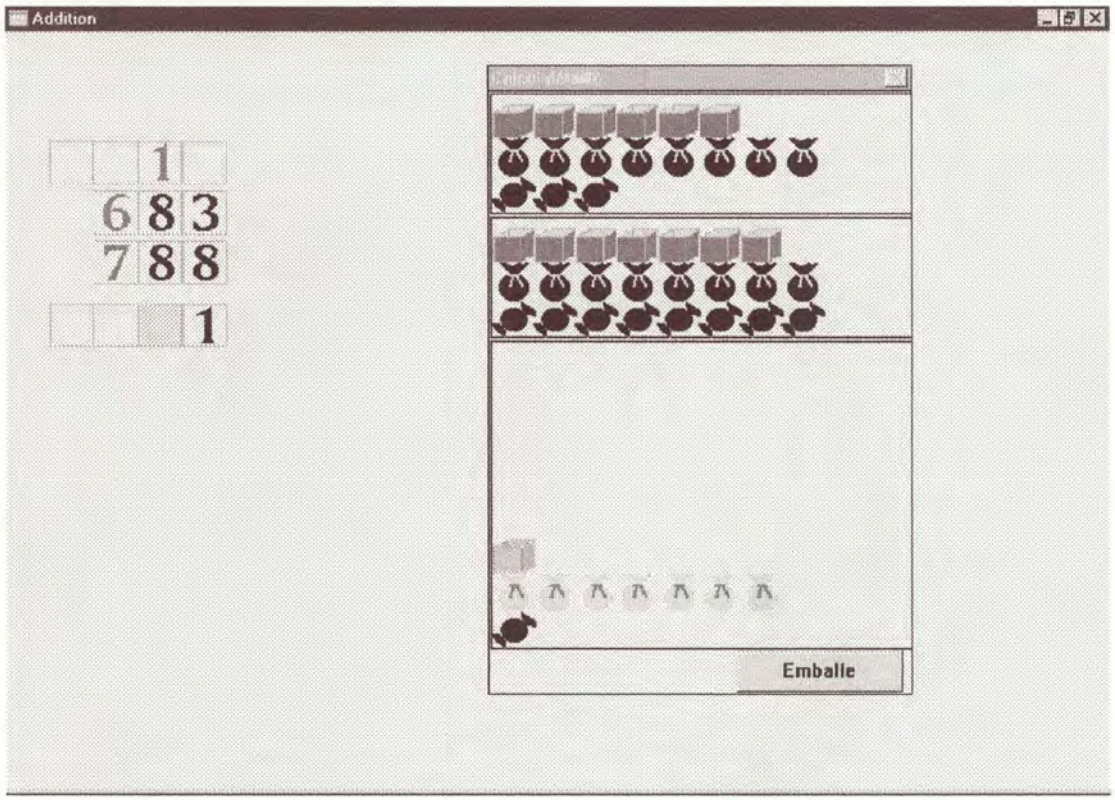
Les sachets du second nombre sont sélectionnés. Il faut maintenant regrouper les sachets par groupe de dix pour obtenir des caisses-reports.



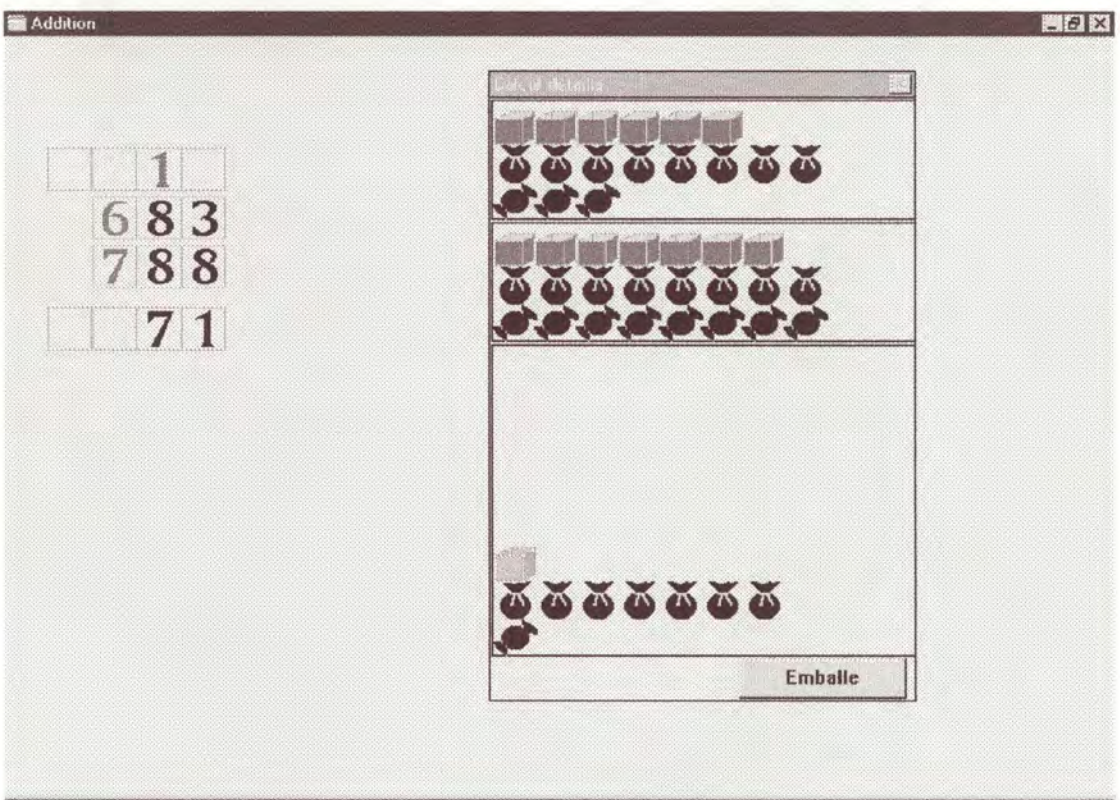
L'élève a sélectionné 10 sachets. Il peut donc enfoncer le bouton « emballe » pour faire apparaître une caisse-report.



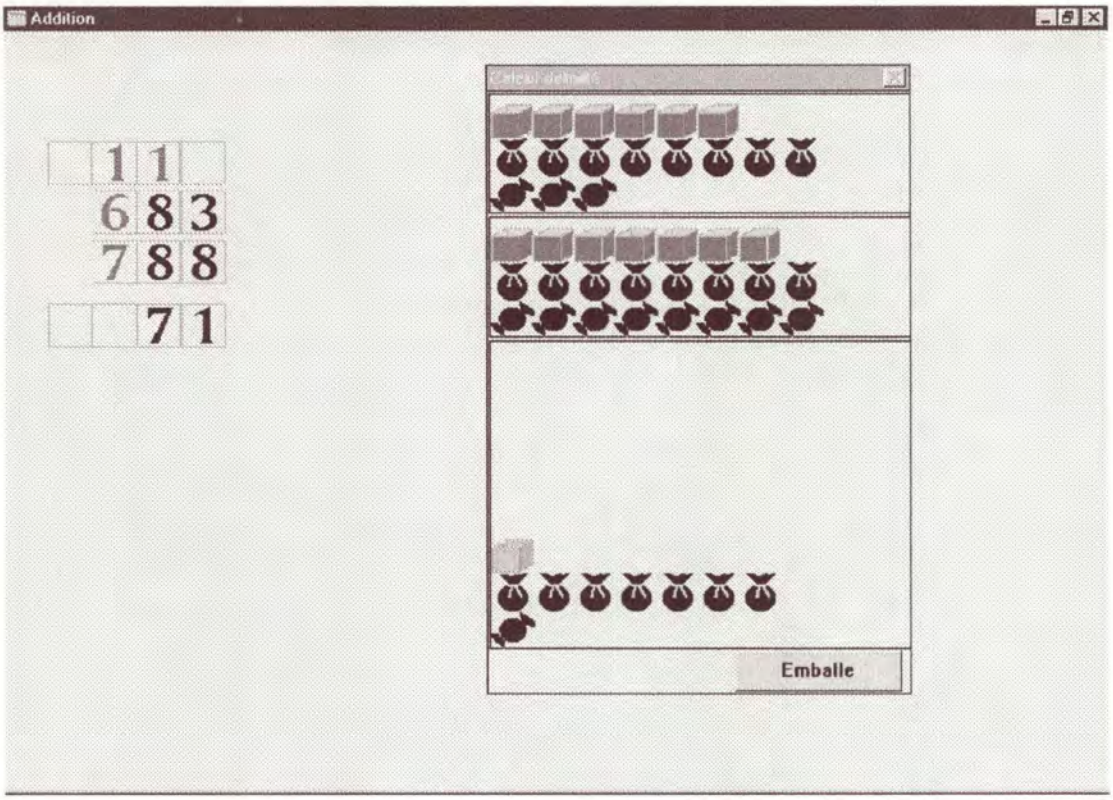
Il reste 7 sachets que l'élève peut sélectionner pour obtenir le résultat au rang des dizaines.



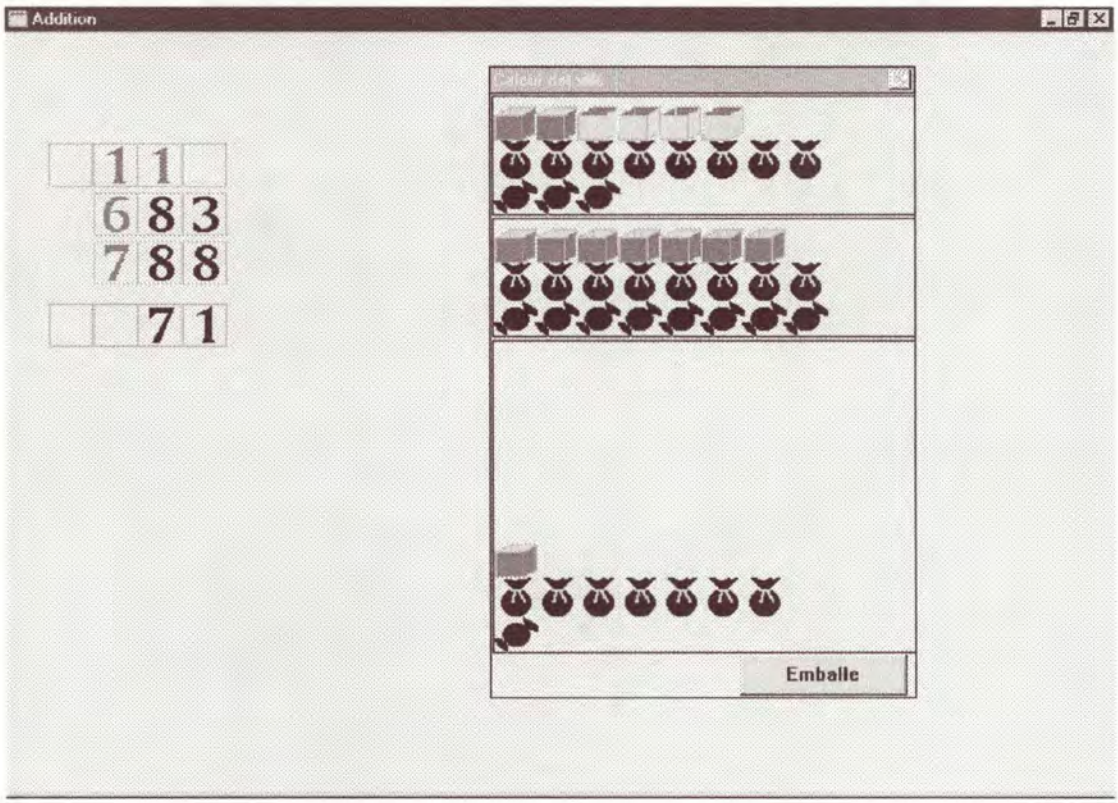
Ici, l'élève a sélectionné la case où il doit inscrire le résultat. Il peut ensuite inscrire le résultat dans cette case. La sélection des sachets dans le calcul détaillé se fait automatiquement.

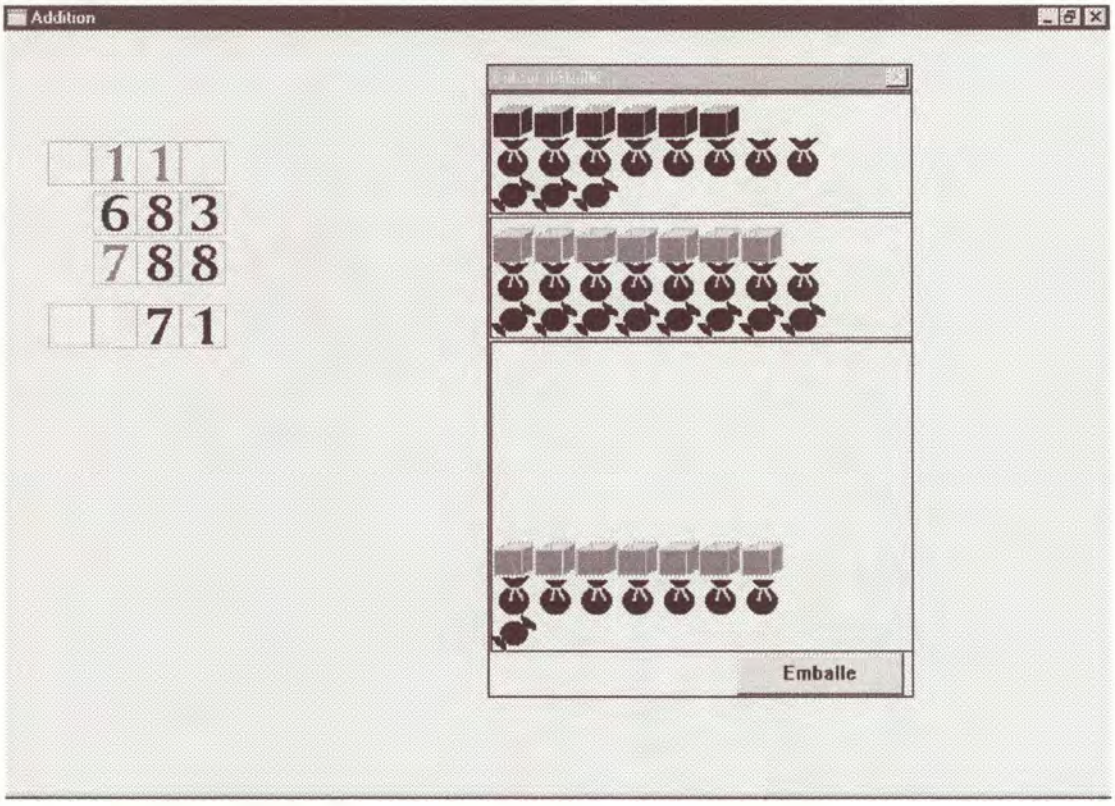


Il peut ensuite inscrire le report au rang des centaines.

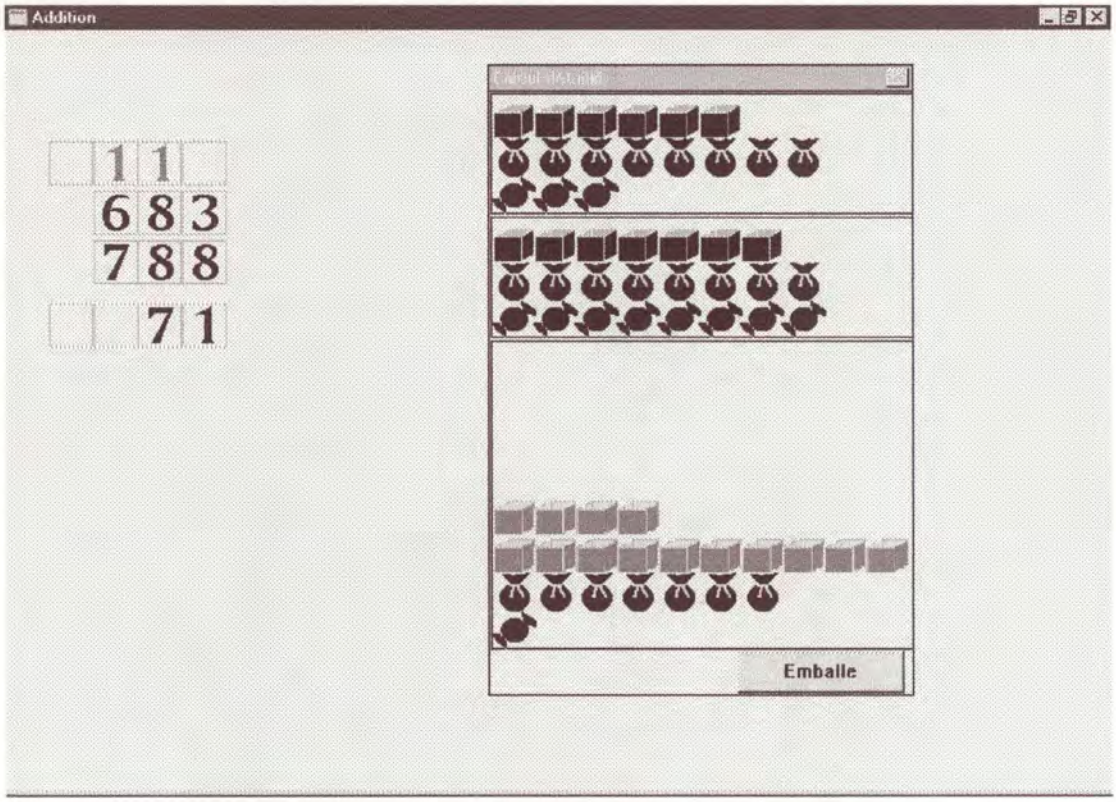


Le rang courant est la centaine. L'élève doit donc sélectionner tous les chiffres de ce rang. Sur l'écran ci-dessous, on peut voir que le chiffre n'est sélectionné que lorsque toutes les caisses le sont. La caisse-report a déjà été sélectionnée.

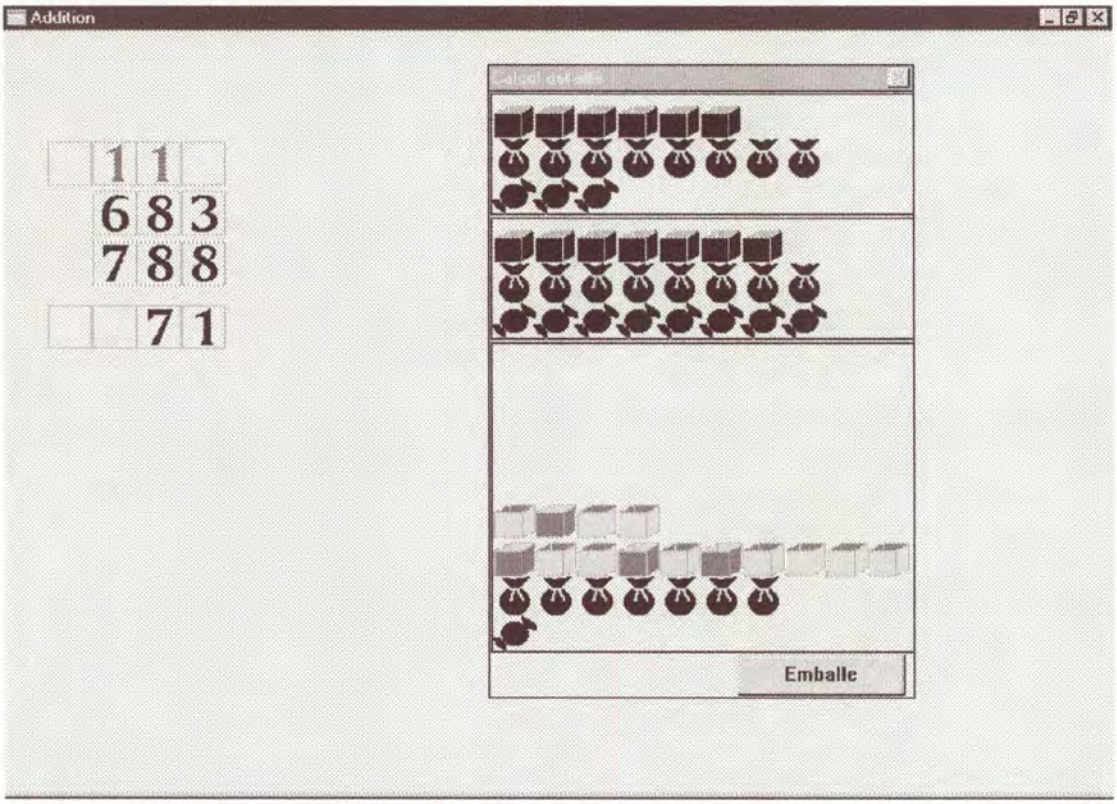




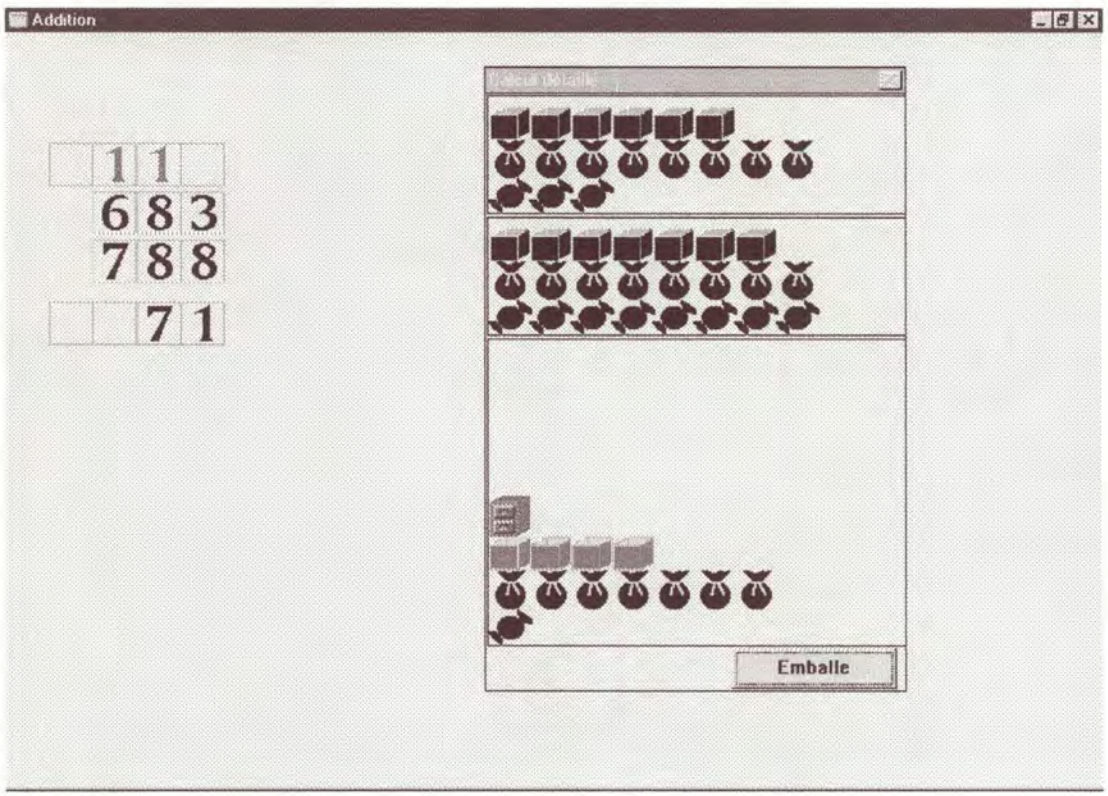
Les caisses du premier nombre sont sélectionnées.



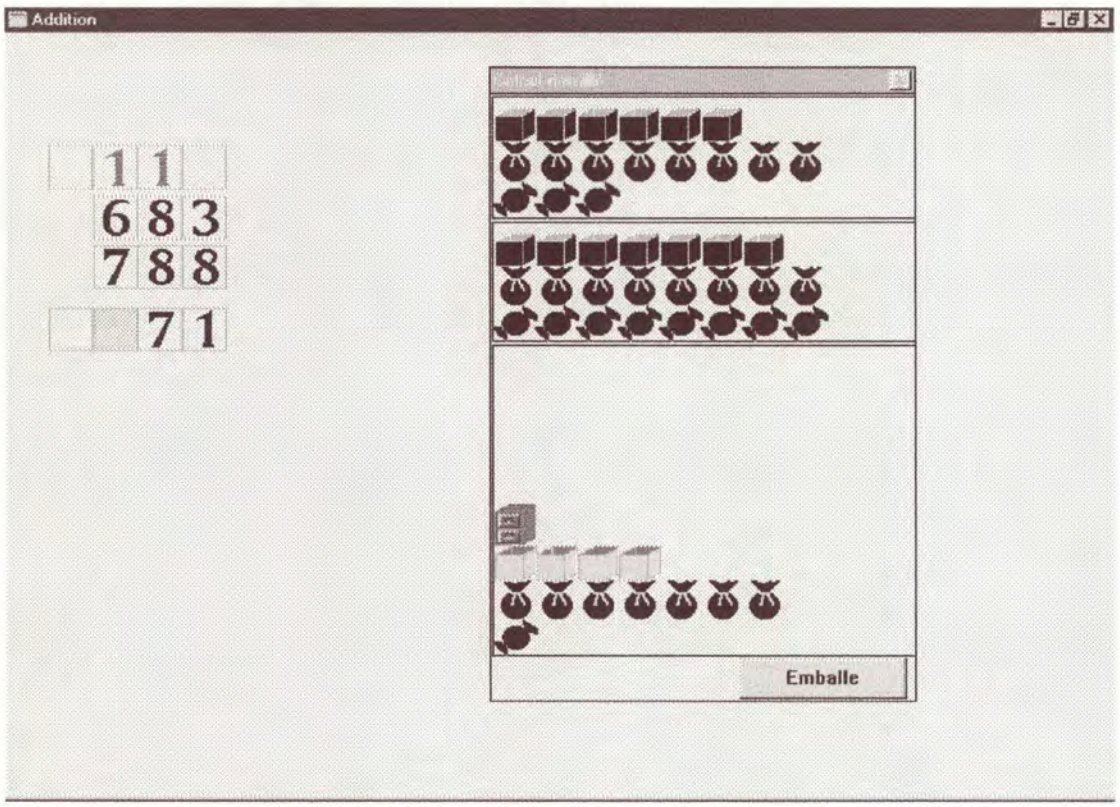
Les caisses du second nombre sont sélectionnées. L'élève peut donc procéder au regroupement de 10 caisses pour obtenir des armoires-reports.



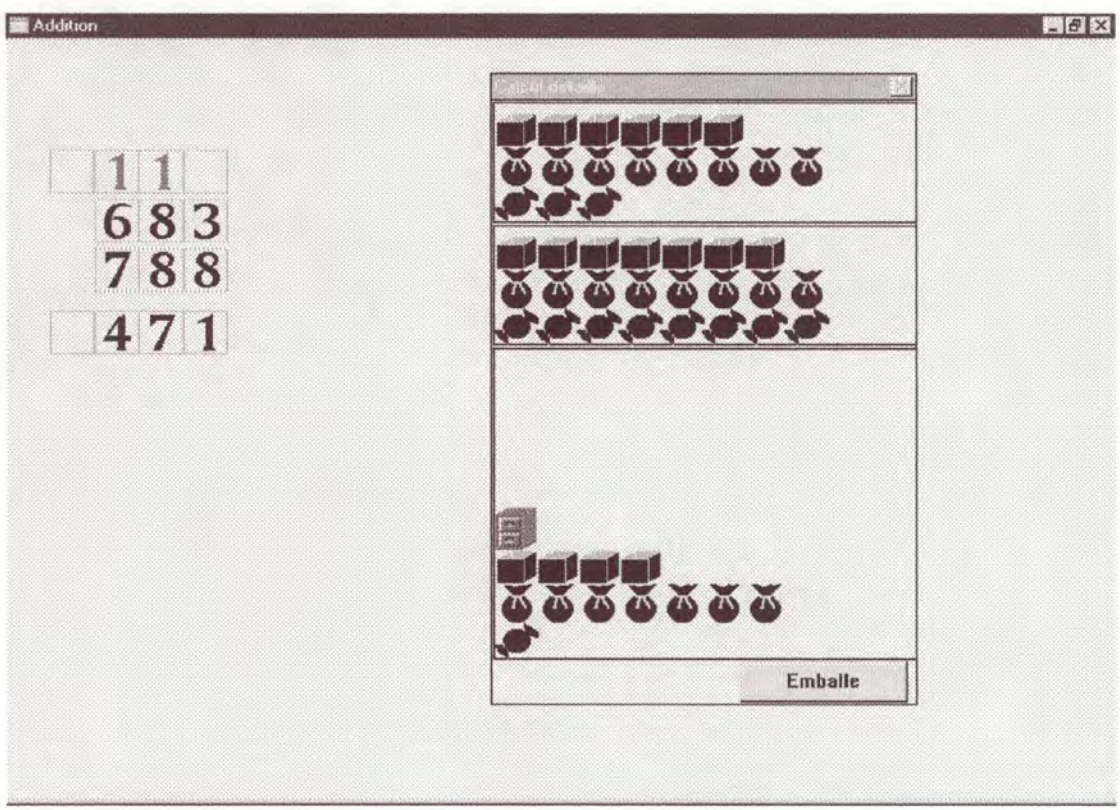
Ici, l'élève a sélectionné dix caisses au hasard. Il peut maintenant enfoncer le bouton « emballe » pour obtenir une armoire-report.



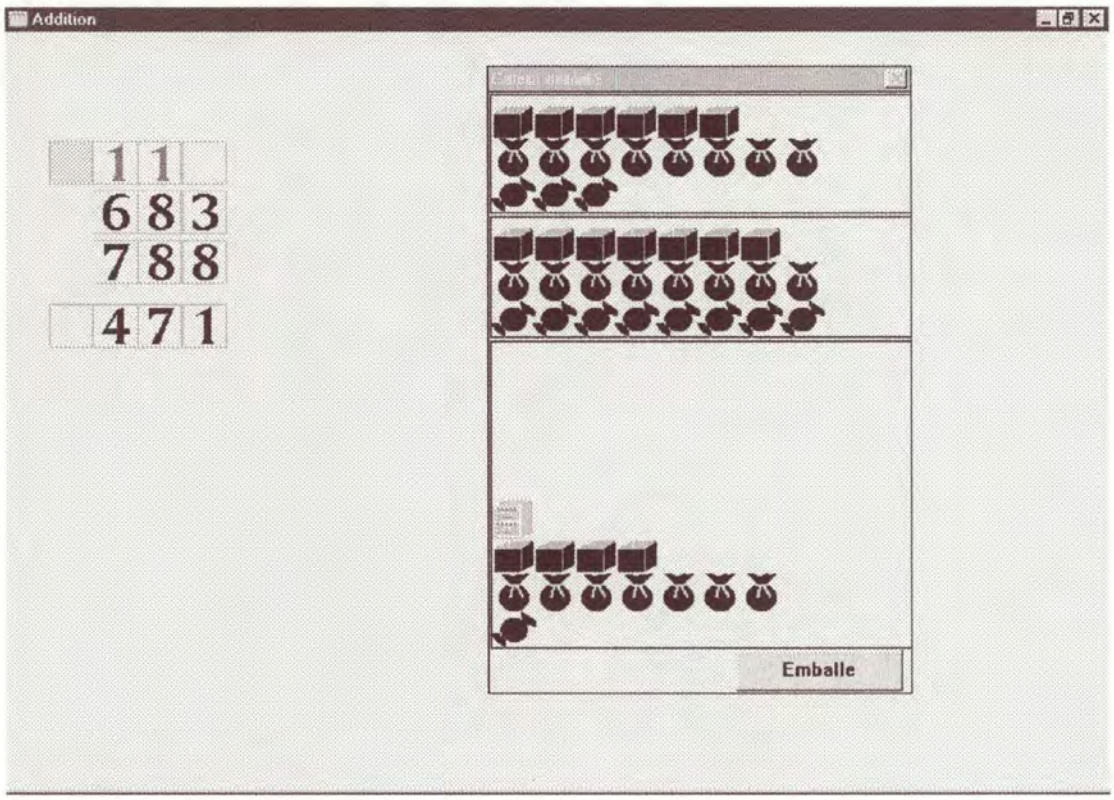
Il reste 4 caisses que l'élève peut sélectionner pour faire apparaître le résultat au rang des centaines. Ici, l'élève a préféré sélectionner la case où il doit inscrire le résultat.

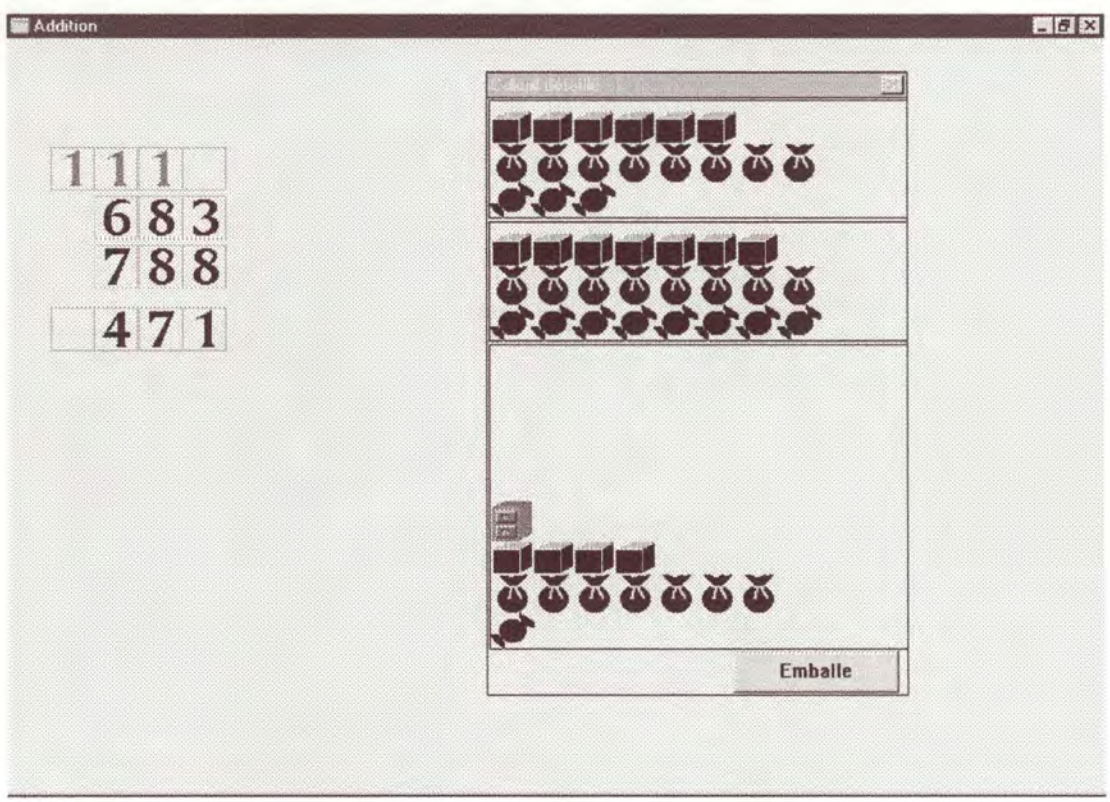


Il inscrit le résultat dans la case des centaines.

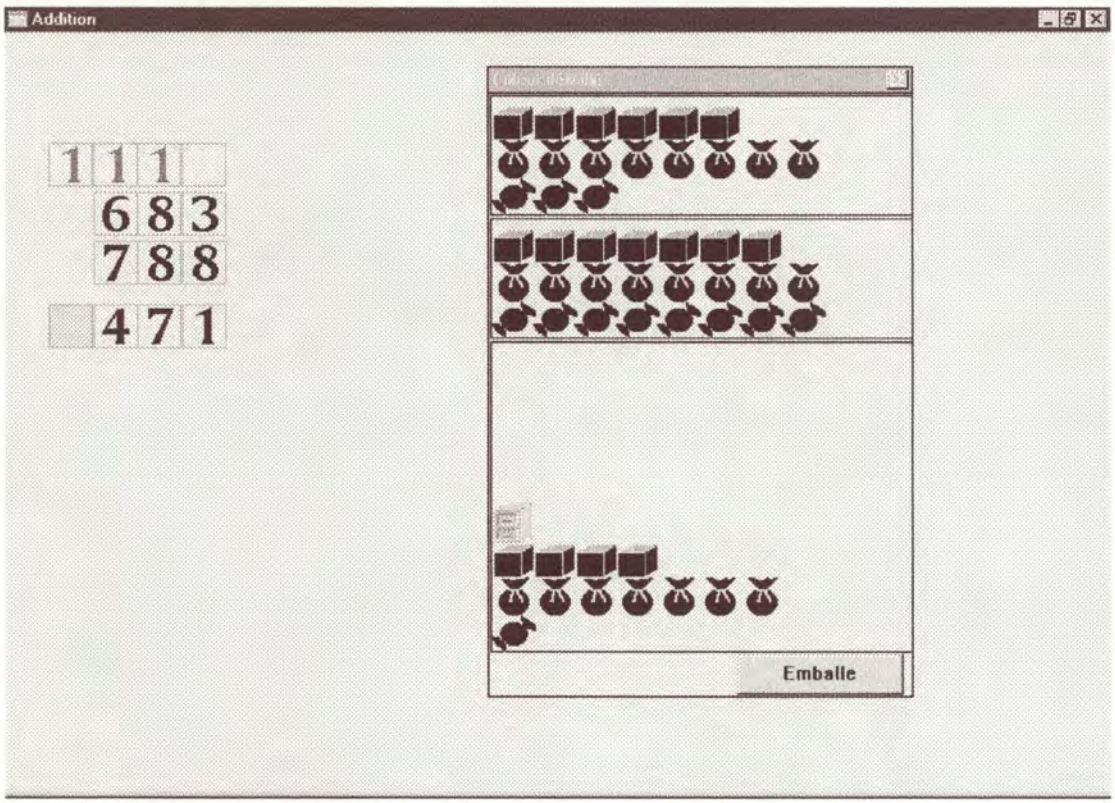


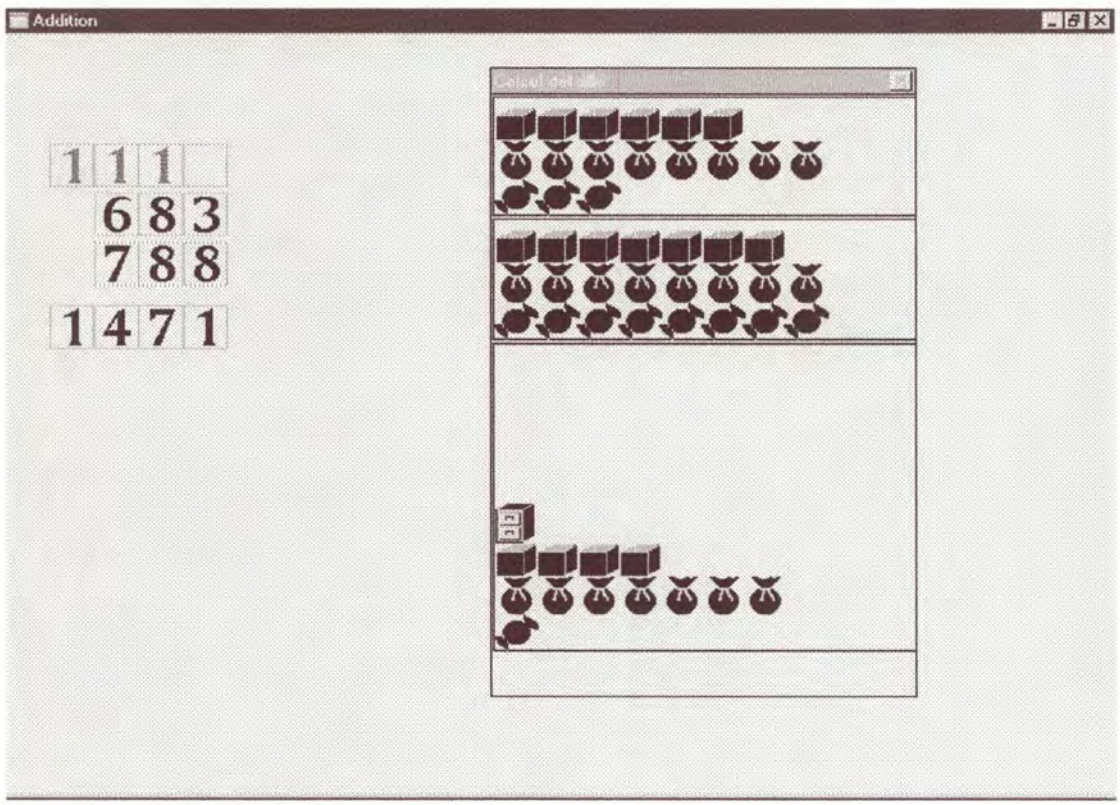
Il peut alors inscrire le report au rang des milliers.





Pour faire apparaître le résultat au rang des milliers, l'élève sélectionne l'armoire-report.





Le résultat apparaît sous forme chiffrée et sous forme imagée dans la fenêtre « calcul détaillé ».

5

CONCLUSION

Ce travail apporte essentiellement deux idées nouvelles. La première est celle de la conception différente des objets de base. Cette conception est plus proche de la réalité. En effet, les « bonbons », « sachets », etc. sont des objets à part entière comme ils le sont dans le monde réel. Il est donc plus facile de leur donner des propriétés. Ces propriétés déterminent notamment les actions que l'on peut effectuer sur eux et celles qui sont interdites. Tous ces objets peuvent de plus être repris pour concevoir ultérieurement d'autres opérations.

La seconde idée est celle du gestionnaire d'événements qui est un premier pas vers le générateur de didacticiels qui a été envisagé dans l'introduction (voir point 1.2.). Il permet aux « non programmeurs » de définir à l'avance le dialogue entre la machine et les élèves qui utiliseront ces logiciels. Pour un didacticiel, la manière dont on répond aux événements qui se produisent tout au long du déroulement d'un programme est très importante. En effet, le but d'un didacticiel est d'apprendre des notions aux élèves et de les aider à comprendre pourquoi il faut procéder d'une telle manière plutôt que d'une autre. C'est dans les explications que l'on donne aux élèves que se trouvent la partie la plus importante d'un didacticiel et la gestion de ces explications est reprise par le gestionnaire d'événements.

Mais il y a encore beaucoup de chemin à parcourir avant d'arriver au générateur de didacticiels espéré. Ce dernier serait l'aboutissement de ce travail et de celui de Marc Philippe. A ces deux travaux pourraient venir se greffer d'autres recherches et d'autres boîtes à outils qui compléteraient l'apprentissage de l'arithmétique fondamentale aux élèves. Un générateur de didacticiel serait le bienvenu pour tous les élèves qui éprouvent des difficultés à apprendre par la manière habituelle. C'est pourquoi il est important de poursuivre les recherches dans ce sens.

6

BIBLIOGRAPHIE

[Biblio 1] : BARON M. et allii, Environnements Interactifs d'Apprentissage avec Ordinateur, Paris, Eyrolles, 1993.

[Biblio 2] : BORLAND INTERNATIONAL, Turbo Pascal / Turbo vision guide, Borland, USA, 1990 .

[Biblio 3] : GIROUX Y., Turbo Pascal pour Windows / Mode d'emploi, Paris, Sybex, 1991.

[Biblio 4] : PHILIPPE Marc., Projet d'aide à la compréhension de l'arithmétique fondamentale, Namur, 1993-1994.

[Biblio 5] : SWAN Tom., Le livre d'or du Borland Pascal 7 / Programmation Windows 3.1, Paris, Sybex, 1993.

7

ANNEXES, CODE

Unit UGlobal ;

Interface

```
uses OBJECTS,Strings,Wintypes,Winprocs,OWindows,ODialogs;
{$R addi.res}

Const

Base = 10 ;
NDecmax  = 0 ;
NChfmax  = 3 ;
L_ico = 32 ;
H_ico = 32 ;
Id_Embal = 1999 ;


Type

intervexpo = -NDecMax.. NChfMax ;
      {intervalle des rangs des chiffres}

possibilite = (NNeg,NPos,Repor,Resul,Error,Sele) ;
      {différentes formes possibles des icônes}

PTablIcon = ^TTablIcon ;
      {Pointeur vers la table des icônes}

TTablIcon = array[possibilite] of HIcon ;
        {table des icônes possibles}

Qualites = (pas_dessi,non_vu,zerocach,sel_ed,sel_able,repo,RepNil,neg,Result,Err) ;
      {différentes caractéristiques d'un icône}

Personnalite = set of Qualites ;
      {ensemble de qualités}

TFChf = array[intervexpo] of integer ;
PFChf = ^TFChf ;

stri20 = record
        case boolean of
        false : (S: string[20] ) ;
        true  : (l : byte ; AZT : array[1..20] of char ) ;
       end ;
```

```
stri  = record
        case boolean of
         false : (S: string ) ;
         true  : (l : byte ; AZT : array[1..255] of char ) ;
        end ;
       {records qui permettent de passer d'un format string pascal à un format AZT}

PRectObj = ^TRectObj ;
TRectObj = Object(TObject)
           private R:Trect ;
           public
           constructor init(Rct:TRect) ;
           procedure SetR(Rct:TREct) ;
           function left: integer ;
           function right:integer ;
           function top : integer ;
           function bottom : integer ;
           function hauteur:integer ;
           function largeur:integer ;
           procedure dessine(DC: HDC; Brush: HBrush) ;
           end ;
       {rectangle capable de se dessiner dans une fenêtre}

function min(a,b:longint):longint ;
function max(a,b:longint):longint ;
       {fonctions qui donnent le minimum et le maximum de deux nombres}

function DivSup(a,b:longint):longint ;
       {arrondit la division vers le haut}

function  CaractStri(c:Qualites) : string ;
function  ChifStri(i:shortint) : string ;
       {fonctions qui donnent les caractéristiques d'une icône et de ses descendants}

procedure ChargeLesIcones(var LesIcones:PTablIcon;n_base:string) ;
Var  Bonbons,Sachets,Caisses,Armoires,Case_vide,plus,fois : PTablIcon ;
    chif : array[0..Base] of PTablIcon ;
       {charge les icônes en mémoire}
```

Implementation

```
var exitsave : pointer ;

function nom_possib(t:possibilite) :string ;
 begin
  case t of
   NNeg: nom_possib:='_NNEG' ;
   NPos: nom_possib:='_NPOS' ;
   Repor: nom_possib:='_REPORT' ;
   Resul: nom_possib:='_RESULTAT' ;
   Error: nom_possib:='_ERR' ;
   Sele: nom_possib:='_inver' ;
  end ;
 end ;
      {donne la terminaison du nom de l'icône dans les ressources}

procedure ChargeLesIcones(var LesIcones:PTablIcon;n_base:string) ;
 var i : possibilite ;
    tampon : stri20 ;
 begin
  for i:= Low(possibilite) to High(possibilite) do
   begin
    tampon.S := N_base+nom_possib(i)+#0 ;
    LesIcones^[i]:=LoadIcon(HInstance,@tampon.AZT) ;
   end ;
 end ;

function DivSup ;
  var Divent,reste : longint ;
 begin
  Divent:=a div b  ; reste:= a mod b ;
   if reste <> 0 then if Divent>0 then inc(Divent) else dec(Divent)  ;
  DivSup:=Divent ;
 end ;

function  CaractStri(c:Qualites) : string ;
 begin
  case  c of
  pas_dessi:CaractStri:=' pas_dessi ;';
  non_vu :CaractStri:=' non_vu ;';
  zerocach:CaractStri:=' à zerocaché ;';
  sel_ed :CaractStri:=' sélectionné ;';
  sel_able :CaractStri:=' selectionnable ;';
  repo  :CaractStri:=' report ;';
  RepNil :CaractStri:=' oper sans report ;';
  neg :CaractStri:=' negatif ;';
  Result :CaractStri:=' Résultat ;';
  Err :CaractStri:=' Erreur ;';
```

```
 end ;
end ;

function  ChifStri(i:shortint) : string ;
 begin
  case i of
   0 :ChifStri := 'ZERO' ;
   1 :ChifStri := 'UN' ;
   2 :ChifStri := 'DEUX' ;
   3 :ChifStri := 'TROIS';
   4 :ChifStri := 'QUATRE' ;
   5 :ChifStri := 'CINQ' ;
   6 :ChifStri := 'SIX' ;
   7 :ChifStri := 'SEPT' ;
   8 :ChifStri := 'HUIT' ;
   9 :ChifStri := 'NEUF' ;
    end ;
 end ;

function min(a,b:longint):longint ;
  begin
   if a<b then min:=a else min:=b ;
  end ;

function max(a,b:longint):longint ;
  begin
   if a>b then max:=a else max:=b ;
  end ;

constructor TRectObj.init(Rct:TRect) ;
 begin
  R:=Rct ;
 end ;

procedure TREctObj.SetR(Rct:TREct) ;
 begin
  R:=Rct ;
 end ;

function TRectObj.left: integer ;
 begin
  left := R.left ;
 end ;

function TRectObj.right:integer ;
  begin
   right:= R.right;
 end ;
```

```
function TRectObj.top : integer ;
 begin
  top := R.top ;
 end ;

function TRectObj.bottom : integer ;
 begin
  bottom := R.bottom ;
 end ;

function TRectObj.hauteur:integer ;
 begin
  with R do hauteur := bottom - top ;
 end ;

function TRectObj.largeur:integer ;
 begin
  with R do largeur := right - left ;
 end ;

procedure TRectObj.dessine(DC: HDC; Brush: HBrush) ;
 begin
  FrameRect(DC,R,Brush) ;
 end ;

procedure UGlobalExit ; far ;
  var i : possibilite ;
      j : byte ;
 begin
  ExitProc := ExitSave ;
  for i:=Low(possibilite) to High(possibilite) do
   begin
    for j:= 0 to base do DestroyIcon(chif[j]^[i]) ;
    DestroyIcon(Case_vide^[i]) ;
    DestroyIcon(Armoires^[i]) ;
    DestroyIcon(Caisses^[i]) ;
    DestroyIcon(Sachets^[i]) ;
    DestroyIcon(Bonbons^[i]) ;
   end ;
  for j:= 0 to base do dispose(chif[j]) ;
  dispose(Armoires) ;
  dispose(Caisses) ;
  dispose(Sachets) ;
  dispose(Bonbons) ;
 end ;
       {procédure de sortie de l'unité, ferme les variables globales
        définies à l'initialisation}
var i : byte ;
```

```
Begin
 ExitSave:=ExitProc ;
 ExitProc:=@UGlobalExit ;
 new(Bonbons) ; ChargeLesIcones(BonBons,'BONBON') ;
 new(Sachets) ; ChargeLesIcones(Sachets,'SACHET') ;
 new(Caisses) ; ChargeLesIcones(Caisses,'CAISSE') ;
 new(Armoires); ChargeLesIcones(Armoires,'ARMOIRE') ;
 new(Case_vide);
 new(Plus) ;
 new(Fois) ;
   for possibilite(i):= Low(possibilite) to pred(High(possibilite)) do
   begin
    Case_vide^[possibilite(i)]:=LoadIcon(HInstance,'RIEN') ;
    Plus^[possibilite(i)]:=LoadIcon(HInstance,'PLUS') ;
    Fois^[possibilite(i)]:=LoadIcon(HInstance,'FOIS') ;
   end ;
    Case_vide^[sele]:=LoadIcon(HInstance,'RIENSEL') ;

 for i:= 0 to base do begin new(chif[i]) ; ChargeLesIcones(chif[i],ChifStri(i)) ;
end;

End .
```

Unit oicone ;

Interface

```
uses OBJECTS,Strings,Wintypes,Winprocs,OWindows,ODialogs, UGlobal;

Type

PIcone = ^TIcone ;
      {objet générique qui permet de représenter une icône dans une case, auquel on peut
      donner des propriétés et sur lequel on peut faire des actions. La représentation
      dépend des propriétés}

TIcone  = Object (TWindow)
       propr : Personnalite ;
       icon  : PTablIcon ;
      constructor Init(AParent:PWindowsObject;X,Y:Integer;
        propri:Personnalite;ico:PTablIcon) ;
      function  GetClassName:PChar ; virtual ;
      procedure GetWindowClass(var AWndClass:TWndClass) ; virtual ;
      procedure SetPr(S,Splus,Smoins:Personnalite;etendue:boolean) ; virtual ;
      function  Selected : boolean ;
      procedure Paint(PaintDC: HDC; var PaintInfo: TPaintStruct); virtual ;
      procedure WMLButtonUp(var Msg:TMessage); virtual wm_First + wm_LButtonUp ;
      procedure WMLButtonDown(var Msg:TMessage); virtual wm_First + wm_LButtonDown ;
      procedure WMRButtonDown(var Msg:TMessage); virtual wm_First + wm_RButtonDown ;
      procedure WMRButtonup(var Msg:TMessage); virtual wm_First + wm_RButtonup ;
      procedure WMMouseMove(var Msg:TMessage); virtual wm_First + wm_MouseMove ;
end ;
```

Implementation

```
var selection:Hicon ;

constructor TIcone.Init(AParent:PWindowsObject;X,Y:Integer;
    propri:Personnalite;ico:PTablIcon) ;
  begin
   inherited init(AParent,Nil) ;
   Attr.style := ws_child or ws_visible ;
   Attr.X:=X ;  Attr.Y:=Y ;  Attr.W:=L_ico ; Attr.H:=H_ico  ;
   propr:=propri ;
   icon :=ico ;
  end ;

function TIcone.GetClassName:PChar ;
  begin
   GetClassName := 'Icone'
  end ;
```

```
procedure TIcone.GetWindowClass(var AWndClass:TWndClass) ;
  begin
   TWindow.GetWindowClass(AWndClass) ;
   with AWndClass do
    begin
     style := style or cs_DblClks or cs_saveBits ;
     hbrBackGround := 0 ;
    end ;
  end ;

procedure TIcone.Paint(PaintDC: HDC; var PaintInfo: TPaintStruct);
  var s:possibilite ;
  begin
   if repo in Propr then  s:= repor else
   if result in Propr then s:= resul else
   if neg in Propr then s:=Nneg else s:=Npos ;
   drawicon(PaintDc,0,0,icon^[s]) ;
   if selected then drawicon(PaintDC,0,0,icon^[sele]) ;
   if err in propr then drawicon(PaintDC,0,0,icon^[error]) ;
  end ;

procedure TIcone.WMLButtonUp(var Msg:TMessage);
  var
     paintDC : HDC ;
     pt,pti:Tpoint ;
  begin
  end ;

procedure TIcone.SetPr ;
  var modif : boolean ;
     tampon : Personnalite ;
  begin
   if Splus + Smoins=[]
    then  tampon:=S
    else  tampon:= propr +Splus -Smoins ;
   if not(tampon=propr) then
    begin
     propr := tampon ;
     InvalidateRect(HWindow,Nil,true) ;
    end;
  end ;

procedure TIcone.WMRButtonup(var Msg:TMessage);
   begin
   end ;
```

```
function TIcone.Selected ;
  begin
   Selected := sel_ed in propr ;
  end ;

procedure TIcone.WMLButtonDown(var Msg:TMessage);
  begin
   if sel_able in propr then
    begin
     if sel_ed in propr
      then  Exclude(propr,sel_ed)
      else begin Include(propr,sel_ed) ; setfocus(Hwindow) end ;
     InvalidateRect(HWindow,Nil,True) ;
    end  else
    begin
    end
  end ;

procedure TIcone.WMMouseMove(var Msg:TMessage);
  begin
   if not selected then
   if Msg.wParam = MK_LButton then
   WMLButtonDown(Msg);
  end ;

procedure TIcone.WMRButtonDown(var Msg:TMessage);
  var s:stri ;
      i : Qualites ;
      rect :TRect ;
  begin
   s.s:=caract ;
   for i:= pas_dessi to Err do if i in propr then S.s:= s.s + CaractStri(i) ;
   S.s:= s.s + #0 ;
   messagebox(0,@s.azt,'propriétes',mb_ok or mb_taskmodal) ;
  end ;

Begin

End .
```

Unit icorang ;

Interface

```
uses OBJECTS,Strings,Wintypes,Winprocs,OWindows,ODialogs, UGlobal,oicone;

Type

Prang = ^Trang ;
TRang   = Object (Ticone)
      r :  intervexpo ;
      associ : PLisRang ;
      constructor Init(AParent:PWindowsObject;X,Y:Integer;
        rang:intervexpo;propri:Personnalite) ;
      destructor Eclate(Rct:TRect) ; virtual ;
      function  nombre:integer ;
      procedure WMRButtonup(var Msg:TMessage);  virtual  wm_first + wm_RButtonup ;
      procedure WMLButtonDown(var Msg:TMessage);  virtual  wm_first + wm_LButtondown;
end ;

PLisRang = ^TLisRang ;
TLisRang = Object (TCollection)
        Cadre:PRectObj ;
        Fenetre:PWindow ;
        dx,dy : shortint ;
        constructor Init(Alimit,ADelta:Integer) ;
        procedure SetPr(S,Splus,Smoins:Personnalite;etendue:boolean) ; virtual ;
        function ToutSel_ed : boolean; virtual ;
        function somme:word ; virtual ;
        destructor done ; virtual ;
        end ;
```

Implementation

```
 constructor Trang.Init(AParent:PWindowsObject;X,Y:Integer;
    rang:intervexpo;propri:Personnalite) ;
  var icone : PTablIcon;
   begin
    r:=rang ;
    associ := Nil ;
    case r of
     0 :icone:= Bonbons ;
     1 :icone:= Sachets ;
     2 :icone:= Caisses ;
     3 :icone:= Armoires ;
     end ;
    inherited init(AParent,X,Y,propri,icone) ;
   end ;
```

```
procedure Trang.WMLButtonDown(var Msg:TMessage);
   begin
    Ticone.WMLButtonDown(Msg);
    if not( associ = nil ) then
     if (PRang(associ^.at(0))^.associ^.ToutSel_ed
     and (PRang(associ^.at(0))^.associ^.count < Base))
     then associ^.SetPr([],[sel_ed],[],false)
     else associ^.SetPr([],[],[sel_ed],false) ;
   end ;

procedure Trang.WMRButtonup(var Msg:TMessage);
   begin
   end ;

destructor Trang.Eclate(Rct:TRect) ;
   var i,j,l,h,tot:integer ;
      desc : Prang ;
   begin
   if r>0 then
   with Rct do
   begin
    l:=((right-left)div L_ico) ;   h:= DivSup((bottom - top), H_ico) ; tot:=1 ;
    for i:=1 to Divsup(10, h) do
    begin j:=1 ;
    while (j<=l) and (tot<=10) do
    begin desc:=new(Prang,init(Parent,left+pred(j)*L_ico,top+pred(i)*H_ico,pred(r),[sel_able]));
          desc^.create; ; inc(tot);inc(j) end ;
    end ;
   done ;
   end else Messagebox(Parent^.HWindow,'Un bonbon ne peut être subdivisé','Attention
          !',mb_Ok) ;
   end ;

function Trang.nombre  ;
  var compte:integer ;
  function memetype(Abox:PWindowsObject):boolean ; far ;
   begin
    if (Typeof(Abox^)=Typeof(Trang)) and (Prang(Abox)^.r=self.r) then inc(compte) ;
   end ;
  begin
   compte:=0 ;
   Parent^.foreach(@memetype) ;
   nombre:=compte ;
  end ;

constructor TLisRang.Init(Alimit,ADelta:Integer) ;
  begin
   Inherited Init(ALimit,ADelta) ;
  end ;
```

```
destructor TLisRang.done ;
  begin
   inherited done ;
  end ;

procedure TLisRang.SetPr ;
  procedure S_Sel(P:Pointer) ; far ;
   begin
    TRang(P^).SetPr(S,Splus,Smoins,etendue) ;
   end ;
 begin
  ForEach(@S_Sel) ;
 end ;

function TLisRang.ToutSel_ed ;
  var Ok : boolean ;
  procedure voirsel(P:Pointer) ;  far ;
   begin
    Ok := Ok and PRang(P)^.Selected ;
   end ;
  begin
   Ok := true ;
   ForEach(@voirsel) ;
   ToutSel_ed:=Ok ;
  end ;

function TLisRang.somme ;
  var total : word ;
  procedure addit(P:pointer) ;  far ;
   begin
    if PRang(P)^.selected then inc(total)  ;
   end ;
  begin
   Total := 0 ;
   ForEach(@addit) ;
   somme:=total ;
  end ;
Begin
End .
```

```
Unit IcoChif;

Interface

uses OBJECTS,Strings,Wintypes,Winprocs,OWindows,ODialogs,Oicone,Uglobal,Icorang,
OFenVisi;

Type

PchiTabl = ^TchiTabl ;
TchiTabl = array [-NDecMax..NchfMax] of Prang ;

PChiffr = ^TChiffr ;
TChiffr = Object (TRang)
      c  :shortint ;
      function norme:shortint ;
      constructor Init(AParent:PWindowsObject;X,Y:Integer;
        rg:intervexpo;propri:Personnalite;vanum:shortint) ;
      procedure setval(vanum:shortint);
      procedure Paint(PaintDC: HDC; var PaintInfo: TPaintStruct); virtual ;
      procedure SetPr(S,Splus,Smoins:Personnalite;etendue:boolean) ; virtual ;
      procedure WMLButtonDown(var Msg:TMessage); virtual wm_First + wm_LButtonDown ;
      procedure WMChar(var Msg:TMessage); virtual wm_First + wm_char ;
      procedure associe(fenetre:PFenVisionDetail;Rct:PRect; e_x,e_y:shortint) ; virtual ;
end ;

PButSpec = ^TButSpec ;
TButSpec = Object (TButton)
          pressed : boolean ;
          procedure WMLButtonDown(var Msg:TMessage); virtual wm_First +
                    wm_LButtonDown ;
         end ;

PChifSom = ^TChifSom ;
TChifSom = Object(TChiffr)
      Et_Av:byte ;
      Liste : PLisChif ;
      Report : PChiffr  ;
      Embal : PButSpec ;
      EmbX,EmbY :Integer ;
      constructor Init(AParent:PWindowsObject;X,Y:Integer;
        rg:intervexpo;propri:Personnalite;vanum:shortint;Li:PLisChif ;Rep:PChiffr;
        Xemb,Yemb:Integer ) ;
      procedure  DebuteOper ;  virtual ;
      procedure  WMTimer(var Msg:TMessage); virtual wm_First + wm_Timer ;
      function Oper_finie : boolean ; virtual ;
      procedure IdEmbal(var Msg:TMessage); virtual id_First + Id_embal ;
      end ;
```

```
PLisChif= ^TLisChif   ;
TLisChif= Object (TLisRang)
        function ToutSEl_ou_Zer : boolean ;  virtual ;
        function somme:word ;  virtual ;
         function Un_Sel_ed(var P:PChiffr):boolean ;
        end ;

Pnombre = ^Tnombre ;
Tnombre = Object (TWindow)
       N:array[intervexpo] of Pchiffr ;
       Procedure MetaZero ;
       Constructor init(AParent:PWindowsObject;x,y:integer;NS:string;
                propri:Personnalite;latent:boolean) ;
       procedure normalise ;
       procedure associe(fenetre:PFenVisionDetail;Rct:TRect) ; virtual ;
       destructor done ;  virtual ;
       function SomChfmod(k:byte;var pivots:PFChf):integer ;
       end ;

PNombFict = ^TNombFict ;
TNombFict = Object (TNombre)
       rang:intervexpo ;
       Constructor Duplique(AParent:PWindowsObject;egal:PNombre;
                propri:Personnalite;rng:intervexpo) ;
       procedure associe(fenetre:PFenVisionDetail;Rct:TRect) ; virtual ;
       end ;

PLisNomb = ^TLisNomb ;
TLisNomb = Object (TCollection)
        function ListeChiffr(rang:intervexpo) : PLisChif ;
        end ;

PNombSom = ^TNombSom ;
TnombSom = Object (TNombre)
       Et_av : byte ;
       Liste : PLisNomb ;
       Report : PNombre  ;
       Constructor init(AParent:PWindowsObject;x,y:integer;propri:Personnalite;
        Li:PLisNomb;Rep:Pnombre;Xemb,Yemb:Integer) ;
       function Oper_finie : boolean ;
       procedure  DebuteOper ;  virtual ;
       procedure  WMTimer(var Msg:TMessage); virtual wm_First + wm_Timer ;
       end ;
```

```
PNombfois= ^TNombfois ;
TNombfois= Object (TNombSom)
        Multiplicande : PNombre ;
        ChfMultiplicateur : PChiffr ;
        Constructor init(AParent:PWindowsObject;x,y:integer;propri:Personnalite;
        M_ateur:PChiffr;M_ande,Rep:Pnombre;Xemb,Yemb:Integer) ;
        procedure  WMTimer(var Msg:TMessage); virtual wm_First + wm_Timer ;
        end ;

PNombProd = ^TNombProd  ;
TNombProd = Object (TNombSom)
         Multiplicande,Multiplicateur : PNombre ;
         Constructor init(AParent:PWindowsObject;x,y:integer;propri:Personnalite;
         M_ateur,M_ande,Rep:Pnombre;Xemb,Yemb:Integer) ;
         procedure  WMTimer(var Msg:TMessage); virtual wm_First + wm_Timer ;
         end ;

PBoite = ^TBoite ;
TBoite = Object (TRang)
      Nba:PNombre ;
      Constructor Init(AParent:PWindowsObject;X,Y:integer;
            rang:intervexpo;propri:Personnalite;sur:PNombre) ;
      Function hminRect :byte  ; virtual ;
      procedure WMLButtonDown(var Msg:TMessage); virtual wm_First + wm_LButtonDown ;
end ;

Var Boi0001,Boi0010{,Boi0100,Boi1000 }: PTablIcon ;
```

Implementation

```
type TR= record  p1,P2 : Tpoint ;
          end ;

constructor TChiffr.Init(AParent:PWindowsObject;X,Y:Integer;
    rg:intervexpo;propri:Personnalite;vanum:shortint) ;
  var icone : PTablIcon ;
  begin
   r:=rg ;
   c:=vanum ;
   icone:=Chif[c] ;
   associ := nil ;
   TIcone.init(AParent,X,Y,propri,icone) ;
  end ;

function Tchiffr.norme:shortint ;
  var norm : shortint ;
  begin
    norm:=c div Base ;
    c:= c mod BASE ;
   if c<0 then
     begin
      c:= BASE + c ;
      dec(norm) ;
     end ;
   norme := norm ;
  end ;

procedure TChiffr.setval(vanum:shortint);
  begin
   c:=vanum ;
   if not(associ=nil) then associe(Nil,Nil,0,0) ;
  end ;

procedure Tchiffr.Paint(PaintDC: HDC; var PaintInfo: TPaintStruct);
    var aspect : stri20 ;
    begin
    if (pas_dessi in propr) or ((c=0) and (zerocach in propr)) then else
     begin
      if non_vu in propr
      then icon:= Case_vide
      else icon:= Chif[c] ;
      Ticone.paint(PaintDC ,PaintInfo);
     end ;
   end ;
```

```
procedure TChiffr.SetPr ;
   begin
    inherited SetPr(S,Splus,Smoins,false) ;
    if etendue and not(associ=Nil)
    then associ^.SetPr(S,Splus,Smoins,etendue) ;
   end ;

procedure TChiffr.associe ;
    var neuf:Prang ;
      i,ecart : shortint ;
  begin
  if associ = Nil then  if Fenetre=Nil then exit else
   begin
    associ:=new(PLisRang,init(9,10));
    associ^.Cadre:= new(PRectObj,init(Rct^)) ;
    associ^.Fenetre := Fenetre ;
    associ^.dx :=e_x ;  associ^.dy :=e_y ;
   end else if not(Rct=Nil) then
    begin
     associ^.cadre^.SetR(Rct^) ;
     associ^.dx :=e_x ;  associ^.dy :=e_y ;
    end ;
 ecart:=abs(associ^.dx) ;
 with associ^ do
 if dx>=0 then
 for i:=count+1 to c do
  begin
   neuf:=new(Prang,
      init(fenetre,Cadre^.left+ (pred(i) mod min(Base,(cadre^.largeur) div dx))*dx+1
      ,Cadre^.bottom-H_ico-dy*(pred(i) div min(Base,(cadre^.largeur) div dx)),r,propr)) ;
      neuf^.associ:=new(PLisChif,Init(1,0)) ;
   neuf^.associ^.insert(@self) ;
   insert(neuf) ;
   neuf^.create ;
  end ;
  with associ^ do while count>c do
  begin
   AtFree(pred(count)) ;
  end ;
 end ;
```

```
procedure TChiffr.WMLButtonDown(var Msg:TMessage);
   begin
   if sel_able in propr then
    begin
    TIcone.WMLButtonDown(Msg);
    if not(associ=nil) then
     if selected then associ^.SetPr([],[sel_ed],[],true)
              else associ^.SetPr([],[],[sel_ed],true) ;
    end ;
   end ;

procedure TChiffr.WMChar(var Msg:TMessage);
  begin
   if selected and (non_vu in propr) then
   if (Msg.wparam>= 48)  and (Msg.wparam<72) then
    begin
     if c= Msg.wparam-48 then SetPr([],[],[non_vu],false) ;
    end ;
  end ;

procedure TButSpec.WMLButtonDown(var Msg:TMessage);
  begin
   inherited WMLButtonDown(Msg) ;
   pressed:=true ;
  end ;

constructor TChifSom.Init(AParent:PWindowsObject;X,Y:Integer;
        rg:intervexpo;propri:Personnalite;vanum:shortint;Li:PLisChif;
        Rep:Pchiffr;Xemb,Yemb:Integer) ;
  begin
   inherited init(AParent,X,Y,rg,propri + [non_vu],vanum) ;
   Et_Av:=0 ; Liste := Li ; Report := Rep ;  embal:=nil ;
   If (rep=Nil) then propr := propr +[RepNil] ;
   EmbX:=Xemb ; EmbY:=Yemb
  end ;

procedure  TChifSom.DebuteOper ;
  begin
   SetTimer(HWindow,1,250,Nil) ;
  end ;
```

```
procedure TChifSom.IdEmbal(var Msg:TMessage);
  begin
   Messagebeep(mb_iconasterisk) ;
  end ;

procedure  TChifSom.WMTimer(var Msg:TMessage);
   var tampon : word ;
      P : PChiffr ;
  begin
   case Et_av of
   0 : begin
       if not(associ=Nil) then
        begin
         embal := new(PButSpec,Init(associ^.fenetre,id_embal,
          ' Emballe ',EmbX,EmbY,120,30,false)) ;
         embal^.pressed := false ;
         Embal^.create ;
        end else embal:=nil ;
       SetPr([],[],[sel_ed,sel_able],true) ;
       Liste^.SetPr([],[Sel_able],[],true) ;
       inc(Et_Av) ;messagebeep(0) ;
      end ;

   1 : if Liste^.ToutSel_ou_Zer and not(Liste^.Un_sel_ed(P))then
       begin
        inc(Et_Av) ; messagebeep(0) ;
        tampon:=liste^.somme ;
        Setval(tampon) ;
        SetPr([],[sel_able],[],true) ;
        if not(embal=nil) then  embal^.pressed := false ;
        end else
       while Liste^.Un_Sel_ed(P) do
        begin
         with P^ do
          begin
           SetPr([],[result],[sel_able,sel_ed],true) ;
           if not(associ=nil) and (repo in propr) then
             with associ^ do
             begin
              if not(cadre=nil) then cadre^.done ;
              done ; associ:=nil ;
             end ;
          end ;
         setval(c+P^.c) ;
        end ;

    2 : if (c<Base) or selected or (embal=nil) then Inc(Et_Av) else
       if embal^.pressed then
        if associ^.somme = Base  then
```

```
    begin
     report^.setval(norme) ;
     setval(c) ; setPr([],[],[sel_ed],true) ;
     inc(Et_Av) ;
    end else embal^.pressed := false ;

3 : if selected or (not(getfocus = HWindow) and (c=0))then
    begin
     if c>=base then
      begin
       if not(report=nil) then report^.setval(norme) ;
       setval(c) ;
      end ;
     if not(getfocus = HWindow) then SetPr([result,sel_ed],[],[],false) ;
     inc(Et_Av) ;
    end ;

4 : if selected and not(non_vu in propr) then
    begin
    SetPr([result],[],[],true) ;
    if not(Report = nil) then
     begin
      report^.SetPr([],[sel_able,repo],[],true) ;
     end ;
     inc(Et_Av) ;
    end ;

5 :  begin if not(report=nil) then
    with report^ do
     if (c=0) or (selected and not(HWindow = GetFocus)) then
       begin
        SetPr([],[sel_ed],[non_vu],false) ;
        inc(Et_Av) ;
       end else
     if selected then inc(Et_Av) else
    else
    inc(Et_Av) ;
    end ;

6 :  if (Report= nil) or
     (report^.selected and not(non_vu in report^.propr)) then
    begin
     if not (Report = Nil)
      then report^.SetPr([],[],[sel_able,sel_ed],true) ;
     liste^.SetPr([],[],[sel_able,sel_ed,non_vu],true) ;
     SetPr([],[],[sel_able,sel_ed,non_vu],true) ;
     inc(Et_Av) ;
    end  ;
```

```
    7 :  begin if not(Embal=Nil) then Embal^.done ; KillTimer(HWindow,1);
         Et_Av:=255;messagebeep(mb_iconasterisk) ; end ;
   end ;
  end ;

function TChifSom.Oper_finie : boolean ;
  begin
  Oper_finie := Et_Av = 255 ;
  end ;

function TLisChif.Un_Sel_ed ;
  function transferable(Item:PChiffr):boolean ;  far ;
   begin with Item^ do
     transferable:= Selected and (Sel_able in propr) and not(c=0)  ;
   end ;
  begin
   P := firstthat(@transferable) ;
   Un_SEl_ed := not(P=Nil) ;
  end ;

function TLisChif.ToutSel_ou_Zer ;
  var Ok : boolean ;
  procedure voirsel(P:Pointer) ;  far ;
   begin
    with PChiffr(P)^ do
     begin
      Ok := Ok and
      (not(sel_able in propr)
       or (zerocach in propr)
       or ((c=0) and ((repo in propr) or not(associ=nil)))) ;
     end ;
   end ;
  begin
   Ok := true ;
   ForEach(@voirsel) ;
   ToutSel_ou_Zer:=Ok ;
  end ;

function TLisChif.somme ;
  var total : word ;
  procedure addit(P:pointer) ;  far ;
   begin
    Total:= total + PChiffr(P)^.c ;
   end ;
  begin
   Total := 0 ;
   ForEach(@addit) ;
   somme:=total ;
  end ;
```

```
procedure Tnombre.MetaZero ;
  var irang : shortint ;
  begin
   for irang:=-NDecMax to NChfMax do
   N[irang]^.c:=0 ;
  end ;


Constructor Tnombre.init(AParent:PWindowsObject;x,y:integer;NS:string;
        propri:Personnalite;latent:boolean)  ;
  var posdecim,i:byte ;
  irang :intervexpo ;
  begin
   inherited init(AParent,Nil) ;
   if latent then Attr.style := ws_child
           else Attr.style := ws_child or ws_visible ;
   Attr.X:=X-L_ico*succ(NchfMax+NDecmax) ;  Attr.Y:=Y ;
   Attr.W:=L_ico*succ(NchfMax+NDecmax) ; Attr.H:=H_ico ;
   for irang:=-NDecMax to NChfMax do
    N[irang]:= new(Pchiffr,init(@self,ATTR.W-L_ico*succ(irang),0,irang,propri,0)) ;
    while ns[1]=' ' do Ns:= copy(Ns,2,length(ns)-1) ;
   posdecim := pos(',',NS) ; if posdecim=0 then posdecim:=length(NS)+1 ;
   for i:=1 to min(length(NS),posdecim+NDecMax) do
    if not(i=posdecim) then
     if i<posdecim then
      with N[posdecim-i-1]^ do
       begin
       c:= byte(NS[i])-48  ;
       SetPr([],[],[non_vu],true) ;
       end else
      with N[posdecim-i]^  do
       begin
       c:= byte(NS[i])-48  ;
       SetPr([],[],[non_vu],true) ;
       end ;
   normalise ;
  end ;

procedure Tnombre.normalise ;
  var chsignif : boolean ;
     irang :intervexpo ;
  begin
   chsignif :=false ;
   for irang:=NChfMax downto -NDecMax do
   with N[irang]^ do
    begin
    if not chsignif then  Chsignif:=not(c=0) ;
    if chsignif then
     begin
```

```
      propr := propr - [Zerocach] ;
     end ;
    end ;
 end ;

procedure Tnombre.associe(fenetre:PFenVisionDetail;Rct:TRect) ;
  var i : shortint ;
      Rt : PRectObj ;
      R : TRect ;
  begin
   R.left := Rct.left+1 ;     R.Top := Rct.top +1 ;
   R.right := Rct.Right-1 ;   R.bottom:= Rct.bottom-1 ;
   Rt:=new(PRectObj,init(R)) ;
   for i:= NChfMax downto 0  do
    begin
     with R do
      begin
      bottom := Rt^.bottom -H_ico*i ;
      top :=bottom-H_ico ;
      end ;
     N[i]^.associe(fenetre, @R,32,32) ;
    end ;
     Fenetre^.insere(Rt) ;
  end ;

function TNombre.SomChfmod(k:byte;var pivots:PFChf):integer ;
  var S:integer ;
      irang : intervexpo ;
  begin
   S:=0 ;
   For irang := -NDecMax to NChfMax do
    begin
     S:=S+DivSup(N[irang]^.c,k)*k ;
     if not(pivots=Nil) then pivots^[irang]:=S ;
    end;
   SomChfmod:=S ;
  end ;

Destructor Tnombre.done ;
  var irang : intervexpo ;
  begin
   for irang := -NdecMax to NchfMax do
    if not(N[irang]=nil) then N[irang]^.done ;
   inherited done ;
  end ;
```

```
Constructor TNombFict.Duplique(AParent:PWindowsObject;egal:PNombre;
                  propri:Personnalite;rng:intervexpo) ;
  var irang,icurrent :intervexpo ;
  begin
   inherited Init(AParent,0,0,'0',propri,true) ;
   Attr.style := ws_child  ;
   rang := rng ;
   for irang := -NDecMax to NChfMax do
    begin
     icurrent:=irang-rang ;
     if (icurrent>-NDecMax)and(icurrent<NChfMax)then
      N[irang]:= new(PChiffr,init(AParent,egal^.N[irang]^.attr.X,egal^.N[irang]^.attr.Y,
          egal^.N[irang]^.r,propri,egal^.N[icurrent]^.c))else
      N[irang]:=new(PChiffr,init(AParent,egal^.N[irang]^.attr.X,egal^.N[irang]^.attr.Y,
          egal^.N[irang]^.r,propri,0)) ;
    end ;
  end ;

procedure TNombFict.associe(fenetre:PFenVisionDetail;Rct:TRect) ;
  var image:PBoite  ;
  begin
   image:=new(Pboite,init(fenetre,Rct.left+32,Rct.top,
            rang,N[0]^.propr+[sel_able]-[sel_ed],@self)) ;
  end ;

Constructor TNombSom.init(AParent:PWindowsObject;x,y:integer;
     propri:Personnalite;Li:PLisNomb;Rep:PNombre;Xemb,Yemb:Integer) ;
  var irang : shortint ;
     Listchif : PLisChif ;
  begin
   TWindow.init(AParent,Nil) ;
   Attr.style := ws_child or ws_visible ;
   Attr.X:=X-L_ico*succ(NchfMax+NDecmax) ;  Attr.Y:=Y ;
   Attr.W:=L_ico*succ(NchfMax+NDecmax) ; Attr.H:=H_ico ;
   Liste := Li ;  Et_Av:=0 ;
   if not(Rep = Nil ) then  Report:= Rep
     else Report:=new(PNombre,init(AParent,x,0,'',[non_vu,repo],false)) ;
   for irang:=-NDecMax to NChfMax do
    begin
     ListChif := Liste^.ListeChiffr(irang) ;
     if irang>0 then ListChif^.insert(Report^.N[irang]) ;
     if irang<NChfMax then
     N[irang]:= new(PchifSom,init(@self,ATTR.W-L_ico*succ(irang)
      ,0,irang,[],0,ListChif,Report^.N[succ(irang)],Xemb,Yemb))
      else
     N[irang]:= new(PchifSom,init(@self,ATTR.W-L_ico*succ(irang)
      ,0,irang,[Repnil],0,ListChif,Nil,Xemb,Yemb)) ;
    end ;
  end;
```

```
function TNombSom.Oper_finie : boolean ;
  begin
   Oper_finie := Et_av= 255 ;
  end ;

procedure  TNombSom.DebuteOper ;
  begin
   SetTimer(HWindow,2,350,Nil) ;
   PChifSom(N[0])^.DebuteOper ;
  end ;

procedure  TNombSom.WMTimer(var Msg:TMessage);
  begin
   case Et_Av of
    0..NChfmax :
              if PChifSom(N[Et_Av])^.Oper_Finie then
               begin
                Inc(Et_Av) ;
                if Et_Av<=NChfMax then
                 begin
                  PChifSom(N[Et_Av])^.DebuteOper
                 end ;
               end ;
    else begin  KillTimer(HWindow,2) ;
           Et_Av:=255 end ;
   end ;
  end ;

function TLisNomb.ListeChiffr(rang:intervexpo) : PLisChif ;
 var list:PLischif ;
 procedure InsereChiffr(P:pointer) ; far ;
  begin
   List^.insert(PNombre(P)^.N[rang]) ;
  end ;
 begin
  list:= new(PLisChif,init(10,0)) ;
  ForEach(@InsereChiffr) ;
  ListeChiffr:=List ;
 end ;

Constructor TNombFois.init(AParent:PWindowsObject;x,y:integer;
     propri:Personnalite;M_ateur:PChiffr;M_ande,Rep:PNombre;Xemb,Yemb:Integer) ;
  var irang : shortint ;
     List : PLisNomb ;
     nbl : PNombFict ;
     i : byte ;
  begin
   Multiplicande := M_ande ;
   ChfMultiplicateur:= M_ateur ;  Et_Av:=0 ;
```

```
  List:=new(PLisNomb,init(10,1)) ;
  For i:=1 to M_ateur^.c do
   begin
    nbl := new(PNombFict,Duplique(AParent,Multiplicande,[sel_able],M_ateur^.r)) ;
    List^.insert(nbl) ;
   end ;
  inherited init(AParent,x,y,propri,List,Rep,Xemb,Yemb) ;
  report^.attr.style := ws_child ;
 end;

procedure  TNombFois.WMTimer(var Msg:TMessage);
 begin
  inherited WMTimer(Msg) ;
 end ;

Constructor TNombProd.init(AParent:PWindowsObject;x,y:integer;propri:Personnalite;
     M_ateur,M_ande,Rep:Pnombre;Xemb,Yemb:Integer) ;
 begin
 end ;

procedure  TNombProd.WMTimer(var Msg:TMessage);
 begin
 end ;

constructor TBoite.Init;
 var irang : intervexpo ;
 begin
  Inherited Init(AParent,X,Y,rang,propri) ;
  case r of
   0:icon := Boi0001 ;
   1:icon := Boi0010 ;
   end;
  nba := sur;
 end ;

Function TBoite.hminRect ;
 var fenmere:PWindow ;
    k : byte ;
    piv : PFchf ;
 begin
  piv:=Nil ;
  fenmere :=PWindow(Parent) ;
  k:=1 ;
  while fenmere^.attr.w < 32*Nba^.SomChfmod(k,piv) do inc(k) ;
  hminRect:=k ;
 end ;
```

```
procedure TBoite.WMLButtonDown(var Msg:TMessage);
  var RC : TRect ;
      fenmere :PFenVisionDetail ;
  begin
   fenmere :=PFenVisionDetail(Parent) ;
   Rc.left :=fenmere^.attr.X ; Rc.top := fenmere^.attr.Y ;
   Rc.right:=Rc.left+fenmere^.attr.w ;
   Rc.bottom:=Rc.top+ hminRect*32 ;
   TNombre(Nba^).associe(fenmere,rc) ;
   done ;
  end ;

BEGIN
 new(Boi0001) ; ChargeLesIcones(Boi0001,'Boi0001') ;
 new(Boi0010) ; ChargeLesIcones(Boi0010,'Boi0010') ;
END.
```

Unit OFenVisi ;

Interface

```
uses OBJECTS,Strings,Wintypes,Winprocs,OWindows,ODialogs,Oicone,Uglobal,Icorang;

Type

PFenVisionDetail = ^TFenVisionDetail ;
TFenVisionDetail = Object (TWindow)
        LisRect : PCollection ;
        constructor Init(AParent: PWindowsObject;
            ATitle: PChar;emb:boolean;xemb,yemb:integer);
        procedure insere(PRect:PRectObj) ;
        procedure Paint(PaintDC: HDC; var PaintInfo: TPaintStruct); virtual;
        destructor Done ; virtual ;
        end ;
```

Implementation

```
constructor TFenVisionDetail.Init;
 begin
  inherited init(Aparent,ATitle) ;
  LisRect := new(PCollection,Init(5,2)) ;
 end ;

procedure TFenVisionDetail.insere(PRect:PRectObj) ;
 begin
  LisRect^.Insert(PRect) ;
 end ;

procedure TFenVisionDetail.Paint(PaintDC: HDC; var PaintInfo: TPaintStruct);
 procedure dessine(P:PRectObj) ; far ;
  begin
   P^.dessine(paintDC,GetStockObject(Black_Brush));
  end ;
 begin
  inherited paint(PaintDC,PaintInfo) ;
  LisRect^.ForEach(@dessine) ;
 end ;

destructor TFenVisionDetail.Done ;
 begin
  LisRect^.done ;
  inherited done ;
 end ;
end .
```

unit UAdditio ;

Interface

```
uses Strings, WinTypes, WinProcs, OWindows,Odialogs ,uglobal, oicone,
            icorang,icochif,OFenVisi;

type

PAddiWindow = ^TAddiWIndow;
TAddiWindow = object(TWindow)
   HasChanged: Boolean;
   ico : Tnombre ;
   tabl : array[0..2] of PNombre ;
   calc_detail:PFenVisionDetail ;
   constructor Init(AParent: PWindowsObject; ATitle: PChar);
   procedure SetupWindow ; virtual ;
   function CanClose: Boolean; virtual;
   procedure WMLButtonDown(var Msg: TMessage);
     virtual wm_First + wm_LButtonDown;
   procedure WMRButtonDown(var Msg: TMessage);
     virtual wm_First + wm_RButtonDown;
end;

var Liste :PLisNomb ;
```

Implémentation

```
constructor TAddiWindow.Init(AParent: PWindowsObject; ATitle: PChar);
var fils:Prang ;
   i,j : byte ;
   va : string[20] ;
   sign : Picone ;
 begin
 fillchar(va,sizeof(va),#0) ;
 inherited Init(AParent, ATitle); Randomize ;
 with attr do style := style ;
 Liste:= new(PLisNomb,init(10,0)) ;
 for i:=1 to 2 do
  begin
   str(random(500)+500:4,va) ;
   for j:=1 to length(va) do if va[j]=' ' then va[j]:='0';
   tabl[i]:=new(PNombre,init(@self,160,80+i*36,va,[zerocach],false)) ;
   Liste^.insert(tabl[i]) ;
  end ;
 sign := new(Picone,init(@self,33,152,[],plus)) ;
 calc_detail:= new(PFenVisionDetail,init(@self,'Calcul Détaillé'
   ,true,180,190));
```

```
 with calc_detail^.attr do
  begin
   style :=  ws_child or ws_visible or ws_border or ws_Caption ;
   title := 'Calcul détaillé' ;
    begin x:=316; y:=0 ; w:=322 ; h:=460 ; end ;
  end ;

 tabl[0]:=new(PNombSom,init(@self, 160,90+3*36,[non_vu],Liste,Nil,180,408)) ;
 with PNombSom(tabl[0])^.report^ do
  begin
   attr.y:=80  ;
  end ;
end;

procedure TAddiWindow.SetUpWindow ;
 var R:Trect ;
     i : shortint ;
 begin   inherited setupwindow ;
  R.bottom:=0;
  for i :=1 to 2 do
  begin
  R.left := 0 ; R.right := calc_detail^.attr.W ;
  R.top := R.bottom  ; R.bottom:= R.top + (3 * succ(H_ico)) ;
  tabl[i]^.associe(calc_detail,R) ;
  end ;
  R.left := 0 ; R.right := calc_detail^.attr.W ;
  R.top := R.bottom ; R.bottom:= R.top + (succ(Nchfmax) *succ(H_ico) ) ;
  PNombSom(tabl[0])^.report^.associe(calc_detail,R) ;
 tabl[0]^.associe(calc_detail,R) ;
  PNombSom(tabl[0])^.debuteoper ;
 end ;

function TAddiWindow.CanClose: Boolean;
var
  Reply: Integer;
begin
  CanClose := True;
end;

procedure TAddiWindow.WMLButtonDown(var Msg: TMessage);
begin
end ;

procedure TAddiWindow.WMRButtonDown(var Msg: TMessage);
begin
end;

begin
end.
```

```
program AddiAppl;

uses Strings, WinTypes, WinProcs, OWindows,Odialogs ,uglobal, oicone,
    icorang,icochif,OFenVisi,UAdditio;

type
 TMyApplication = object(TApplication)
   procedure InitMainWindow  ; virtual;
 end;


procedure TMyApplication.InitMainWindow;
begin
 MainWindow := New(PAddiWindow, Init(nil, 'Addition'));
end;

var
 MyApp: TMyApplication;

begin
 MyApp.Init('Addition');
 MyApp.Run;
 MyApp.Done;
end.
```