MASTER EN SCIENCES INFORMATIQUES

Implémentation d'un didacticiel général d'aide à la mémorisation

Walthéry, Valérie; Vignaux, Philippe

Award date:
1994

Awarding institution:
Universite de Namur

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

Rue Grandgagnage, 21, B-5000 Namur (Belgium)

Année académique 1993-1994

Implémentation d'un didactitiel général d'aide à la mémorisation

Mémoire de fin d'études présenté par

Valérie Walthéry et Philippe Vignaux

en vue de l'obtention du grade de licencié et maître en informatique.

Promoteur: Claude Cherton.

ABSTRACT

Ce mémoire propose un didacticiel général d'aide à la mémorisation. Dans un premier temps, une théorie relative au fonctionnement de la mémoire humaine est exposée, celle du système humain de traitement de l'information, ainsi que l'une de ses variantes, la théorie duplex. Ensuite, les différentes parties du logiciel, les concepts et les fonctionnalités utilisés dans celui-ci sont présentés. Les fonctionnalités sont alors analysées en terme d'objet et de spécification. Une justification des choix de programmation et un relevé des améliorations potentielles termineront le travail. Le code du logiciel est disponible en annexe.

This thesis proposes a general educational software program which helps memorisation. In a first part, a theory about the functioning of human memory will be set out, the one about the human system of information treating as well as one of its variations, the duplex theory. After that, the different parts of the software, the concepts and the functionalities used in this one will be expounded. The functionalities are then analysed in term of object and specification. The thesis will end with a justification of the programming choices and a listing of the potential improvements. The software code is available in the annexes.

Nos remerciements s'adressent tout d'abord à notre promoteur, Monsieur Claude Cherton, pour la disponibilité dont il a fait preuve ainsi que pour l'assistance qu'il nous a fournie tout au long de notre mémoire. Nous laissant continuellement l'initiative, il a su susciter en nous les questions pertinentes.

Notre reconnaissance va également à Monsieur Donnay, de la Fondation Travail Université, pour les livres fournis sur le sujet. Ceux-ci nous furent de la plus grande utilité. De même, que soient remerciées les personnes qui ont témoigné de l'intérêt pour notre travail en nous procurant de la documentation.

Nos pensées vont également vers nos familles et nos amis qui n'ont cessé de nous encourager et de nous soutenir.

Enfin, nous remercions vivement les personnes qui ont accepté de tester le logiciel et qui ont proposé des avis pertinents sur celui-ci. De même, nous exprimons toute notre gratitude à Monsieur Jean-Marie Denis, qui a assuré la correction orthographique du texte de ce mémoire.

Mise en page : Valérie Walthéry

TABLE DES MATIERES

1

INTRODUCTION

Par Valérie Walthéry

La mémoire est un domaine très vaste, capable d'assimiler une grande variété d'informations utiles ou inutiles, par plaisir ou par nécessité. La vie quotidienne y recourt constamment, que ce soit pour retenir des numéros de téléphone, la liste des courses à faire ou la date et l'heure d'un rendez-vous. Il en est de même en ce qui concerne le travail : les étudiants doivent, par exemple, mémoriser des cours, du vocabulaire; les médecins doivent connaître les symptômes des maladies, les propriétés des médicaments.

Toutes les informations mémorisables ont une structure. Du vocabulaire français - anglais peut, par exemple, se représenter sous la forme d'une liste de mots à deux colonnes, la première contenant les mots en français, la seconde contenant les mots en anglais. Une liste de courses à faire peut soit se représenter sous la forme d'une liste à une seule colonne, soit être plus structurée et comporter plusieurs catégories : une pour les légumes, une pour les boissons,... La plupart des exemples cités ci-dessus suivent ces structures de listes de mots ou de nombres à une ou plusieurs colonnes. Par contre, l'exemple des cours d'un étudiant ne rentre pas dans cette catégorie.

En effet, donner à un texte une structure de liste de mots à plusieurs colonnes n'est pas pratique : la signification des phrases risque de se perdre ou de causer des difficultés quant à la représentation. Nous avons donc dégagé deux types d'informations différents : les listes de mots ou de nombres et les textes.

Certaines méthodes d'aide à l'amélioration de la mémorisation proposées dans les différents livres consultés (Biblio 1-4-5) s'appliquent aux listes de mots et de nombres, d'autres aux textes. Par exemple, la méthode de l'élaboration progressive (Biblio 4, pages 208 à 210), associant la liste de mots à un itinéraire précis et bien connu du sujet, s'applique parfaitement au type d'information mis sous forme de liste, mais ne convient pas du tout au type texte. En ce qui concerne l'aide à la mémorisation des textes, nous n'avons trouvé que des petites astuces qui permettent, par exemple, de retenir le titre d'un texte ou la synthèse de celui-ci.

L'immensité du sujet « Mémoire » et l'existence de nombreux exercices impose des choix précis. C'est pourquoi nous avons étudié les trois stratégies :

- étude des méthodes d'aide à la mémorisation des listes uniquement;
- étude des méthodes d'aide à la mémorisation des textes uniquement;
- développement des méthodes d'aide à la mémorisation des listes et des textes sans approfondissement des deux types.

L'analyse démontre que, des trois stratégies, seule la première mérite d'être retenue, pour trois raisons :

- premièrement, les listes couvrent un domaine très large. Il est possible de mettre une structure de liste sur beaucoup de types d'informations, ce qui s'avère plus difficile pour les textes. En effet, pour structurer des textes efficacement, il faudrait, par exemple, utiliser la technique des réseaux sémantiques. Celle-ci exigerait un mémoire à elle seule;

- deuxièmement, les méthodes permettant de faciliter ou d'améliorer la mémorisation des listes sont plus précises et, par conséquent, plus facilement spécifiables que celles aidant à la mémorisation d'un texte. Les premières conviennent donc mieux à une informatisation. De plus, les méthodes d'aide à la mémorisation de textes demandent, généralement, un gros effort d'imagination de la part des sujets : ils doivent, par exemple, se raconter une histoire qui illustre le texte à apprendre; l'ordinateur est incapable de faire cet effort d'imagination à la place du sujet.

- troisièmement, il existe, dans les livres consultés, beaucoup plus d'exercices liés à la mémorisation des listes qu'à la mémorisation des textes.

Après avoir décidé de développer les listes de mots et de nombres, nous nous sommes interrogés sur le processus de mémorisation d'une liste de mots. Nous avons remarqué que nous procédions, tous les deux, de la même manière. Les différents points communs de la méthode utilisée sont les suivants :

- lorsque la liste de mots à apprendre est longue, nous l'étudions par petites sections;
- lorsqu'un petit groupe de mots est connu, nous le répétons afin de ne pas l'oublier immédiatement;
- lorsqu'un mot est difficile à retenir, nous utilisons un artifice pour faciliter la mémorisation, c'est-à-dire que nous nous donnons une représentation imagée et mentale de ce mot;
- lorsque l'occasion se présente, nous groupons les mots en plusieurs catégories. Les mots d'une même catégorie ont, en effet, un rapport entre eux.

La plupart des techniques permettant de rendre la mémorisation des listes plus efficace reprennent souvent des concepts et des conseils identiques à ceux formulés ci-dessus. Nous en avons dégagé trois qui nous paraissent importants et intéressants. Les méthodes d'apprentissage des listes que nous avons développées reposent sur ces trois concepts, dont voici une brève explication :

- la répétition : répéter plusieurs fois les même mots permet de les ancrer plus rapidement dans la mémoire (voir chapitre 2.1);

- les associations ou images mentales : associer un mot à un ou plusieurs autres mots connu(s) par ailleurs ou illustrer mentalement le mot à mémoriser constitue une aide à la mémorisation (voir chapitre 3.3.1.4);

- les groupements : ils servent à mieux structurer les listes. Ce qui est bien structuré est plus aisément retenu (voir chapitre 3.3.1.4).

En plus de ces trois concepts de base, la méthode d'apprentissage des listes que nous avons développée permet de diviser la liste à apprendre en plusieurs sections (voir chapitre 3.3.2.1).

Après avoir expliqué la cause du choix des listes comme partie centrale de ce mémoire et introduit les concepts de base, il reste à préciser la signification du mot « liste », notion très vague en soi. Il pourrait s'agir de listes telles qu'elles sont définies en informatique (arbres,...), ou de simples listes linéaires non structurées (simples suites de mots). Nous avons préféré définir un autre type de liste, plus complexe que les listes linéaires mais moins général que les arbres : les listes à deux niveaux. Il s'agit de listes de listes de mots ou de nombres, ou, pour être plus précis, de listes d'items dont les items sont des listes linéaires de mots ou de nombres.

Cet exemple permet de mieux situer les listes à deux niveaux. La liste suivante des courses à faire comporte sept mots :

Rôti
Haricots
Eau
Vin
Salade
Steaks
Tomates

Elle peut être mieux structurée et mise sous la forme d'une liste à deux niveaux :

Rôti	Steaks	
Haricots	Salade	Tomates
Eau	Vin	

Cette nouvelle liste contient trois items : le premier est une liste de deux mots, le deuxième est une liste de trois mots et le troisième est une liste de deux mots. Nous avons donc bien une liste d'items dont les items sont des listes linéaires de mots. Nous définirons avec plus de précision les listes au chapitre 3.3.1.1.

Comme nous l'avons signalé plus haut, l'aide à la mémorisation des listes à deux niveaux constitue la partie essentielle de ce mémoire. Néanmoins, deux tests, rencontrés dans le livre de Gauquelin (Développer sa mémoire : méthode Richaudeau, Biblio 4), permettent de préparer et d'aider à l'apprentissage des listes.

Le premier test donne des indications intéressantes sur nos modes de mémorisation privilégiés, des conseils sur la façon d'améliorer cette mémorisation. Le second test détermine notre empan de mémoire : il donne une idée de la capacité de notre mémoire. En recommençant ce second test plusieurs fois, il est possible de parvenir à améliorer cette capacité. C'est pourquoi nous parlerons parfois d'exercice de l'empan en plus de test de l'empan. Ces deux tests sont repris dans notre travail. Nous conseillons vivement de les exécuter avant de commencer l'apprentissage d'une liste quelconque de mots ou de nombres. Nous détaillerons ces deux tests respectifs aux chapitres 3.1 et 3.2.

2

APPROCHE THEORIQUE DE LA MEMOIRE

Par Valérie Walthéry

Les ouvrages que nous avons consultés sont de deux types : les uns exposent des théories, décrivent la mémoire, ses composantes et la manière dont elle fonctionne; les autres expliquent des techniques qui permettent de développer, d'entretenir la mémoire, ou de faciliter la mémorisation de certains types d'informations.

Dans un premier temps, nous définirons quelques concepts relatifs à la mémoire et exposerons une théorie sur son fonctionnement. Ensuite, nous parlerons des techniques qui aident et facilitent la mémorisation.

2.1. Définition et mode de fonctionnement de la mémoire

Il n'existe pas une mais plusieurs théories ou modèles qui tentent d'expliquer le fonctionnement de la mémoire. Nous exposons ici la synthèse d'un modèle de la mémoire appelé « système humain de traitement de l'information ». Ce modèle est présenté en détail dans le livre de R.L. Klatzky (Human memory, structures and processes, Biblio 2). Nous exposons ensuite une variante à ce modèle : « la théorie duplex ».

2.1.1. Le système humain de traitement de l'information

La figure II.1, ci-dessous, décrit le fonctionnement du système humain de traitement de l'information lorsqu'il reçoit un stimulus provenant du monde réel. Nous allons expliquer le chemin suivi par les informations à travers le système afin d'illustrer le modèle :

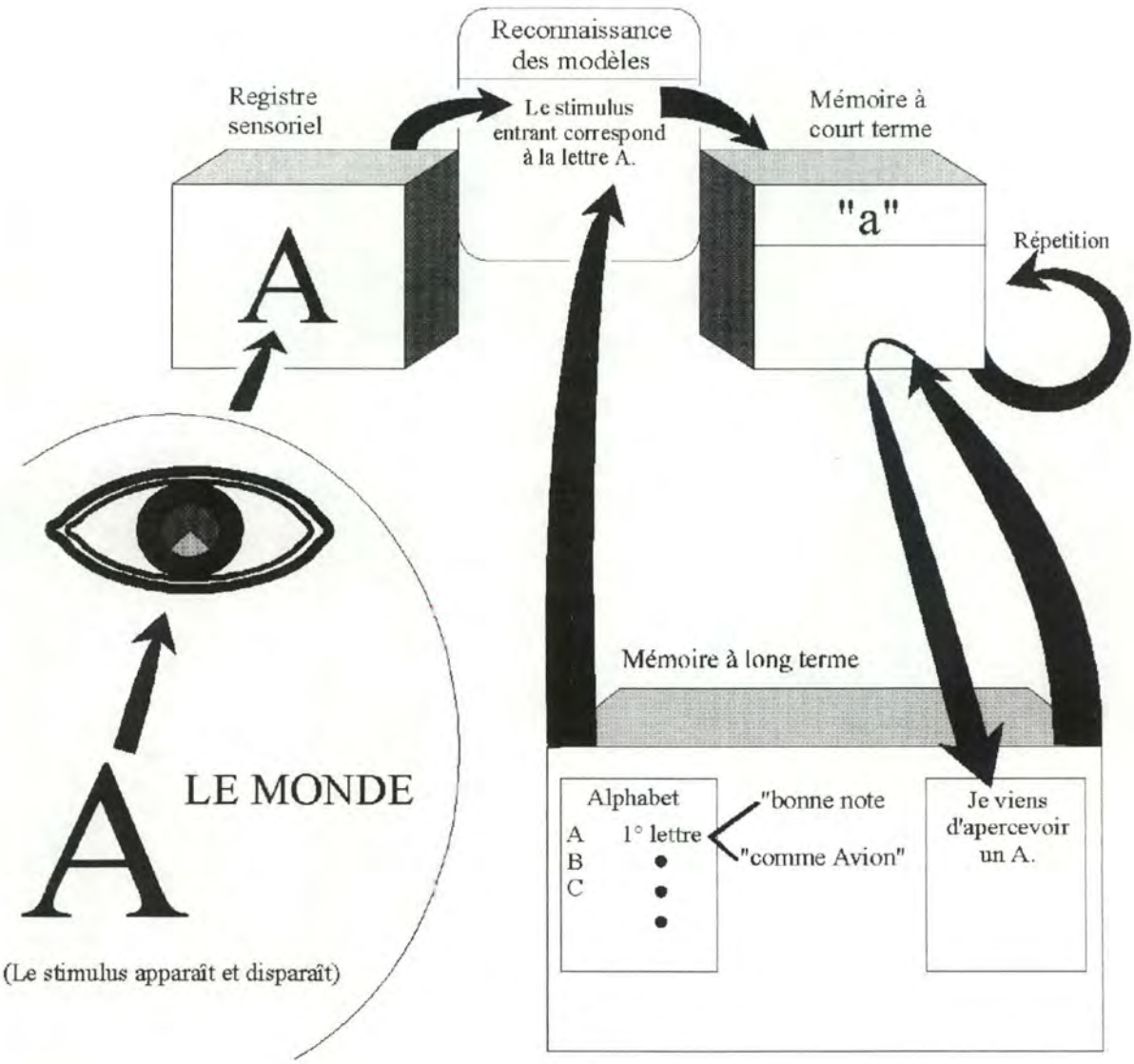


FIGURE II.1

Le système humain de traitement de l'information est divisé en trois structures de stockage principales; chacune d'entre elles correspond à un niveau différent de traitement de l'information.

Lorsqu'un stimulus est présenté au système (sur la figure II.1, il s'agit de la lettre « A »), une certaine quantité d'information concernant ce stimulus entre dans le système par le premier niveau de traitement de l'information appelé « registre sensoriel » parce que les informations qui y sont stockées rentrent dans le système par l'intermédiaire d'un ou de plusieurs sens et qu'elles y sont gardées un court instant sous cette forme sensorielle (un son est, par exemple, gardé sous sa forme sonore, auditive). Il y a donc un registre sensoriel pour chaque sens.

Les informations restent dans ces registres très peu de temps. Plus elles y restent, plus elles deviennent floues, jusqu'à disparition complète. Cet affaiblissement progressif est appelé « extinction ».

Pendant que les informations concernant le stimulus se trouvent dans un registre sensoriel, avant qu'elles ne passent à la structure de stockage suivante, c'est-à-dire à la mémoire à court terme, un processus complexe et important entre en jeu : la reconnaissance des modèles. Ce processus résulte de la comparaison des informations stockées dans les registres sensoriels et des connaissances acquises précédemment par le système. Un modèle est dit « reconnu » lorsque l'aspect sensoriel des informations est mis en relation avec des concepts connus et sensés.

Pour en revenir à notre exemple, quand la lettre « A » entre dans le registre visuel, elle y est stockée comme un ensemble de lignes attachées par certains points et orientées d'une certaine manière. Le processus de reconnaissance des modèles met cet ensemble en relation avec un concept connu : la lettre « A ». Un sens plus spécifique de la reconnaissance des modèles est donc de donner un nom au stimulus. Ce n'est pas toujours possible, ce qui n'empêche pas de pouvoir reconnaître celui-ci. La reconnaissance des modèles permet donc d'encoder le stimulus de manière à ce qu'il puisse passer au deuxième niveau de traitement du système : « la mémoire à court terme (MCT) ».

Contrairement aux registres sensoriels, la mémoire à court terme ne stocke pas les informations sous leur forme primitive, sensorielle. Dans notre exemple, la lettre « A » est retenue non pas comme un ensemble de lignes, mais bien comme une lettre, A.

De plus la mémoire à court terme est capable de conserver des informations plus longtemps que les registres sensoriels. Un item du registre visuel s'éteint plus ou moins en une seconde tandis qu'il peut être maintenu indéfiniment en mémoire à court terme. En effet, un processus de répétition permet de « recycler » continuellement l'item à travers la mémoire à court terme de manière à ce que l'extinction de cet item ne soit jamais complète. Sans ce processus de répétition, les informations contenues dans la mémoire à court terme sont perdues après un temps proche de plus ou moins trente secondes.

La mémoire à court terme est limitée pour une deuxième raison : seuls quelques stimuli peuvent y être stockés simultanément (même avec l'aide de la répétition), ainsi que le démontre l' « expérience de l'empan » ou « expérience de la mémoire immédiate ». Dans cette expérience, un sujet reçoit une courte liste d'items (des mots, par exemple) et doit la répéter immédiatement après l'avoir lue. Les résultats de cette expérience montrent que les performances des sujets sont parfaites tant que la liste présentée ne dépasse pas sept items au maximum. A partir de sept items, la répétition devient erronée. Par conséquent, la mémoire à court terme est disponible pour sept items seulement.

Cette limite est appelée « empan de mémoire ». Les sept items peuvent être maintenus en mémoire à court terme le temps nécessaire à leur répétition. Il suffit qu'un nombre trop important de stimuli soient présentés en même temps pour qu'une certaine quantité soit perdue. Cette perte est une forme d'oubli. Le mot « oubli » est employé pour se référer à une perte d'informations à partir de n'importe quel niveau de stockage du système.

La troisième structure de stockage du système humain de traitement de l'information est « la mémoire à long terme (MLT) ». Ce lieu de stockage est permanent et essentiel pour toutes les connaissances acquises sur le monde. La mémoire à long terme poursuit deux buts. Premièrement, après que le stimulus ait été reconnu et soit entré en mémoire à long terme, il peut être retenu de façon plus permanente par cette mémoire à long terme.

Deuxièmement, les connaissances que la mémoire à long terme contient sont cruciales pour le processus de reconnaissance de modèles. En effet, ces connaissances sont mises en relation avec les informations entrant dans le système de traitement de l'information dans le but de les reconnaître.

Un dernier point à noter à propos du système est qu'il y a des limitations à notre capacité de traitement des informations, notamment sur le temps de stockage et sur la quantité d'information que l'on peut traiter simultanément. Cette limitation de capacité apparaît à tous les niveaux du système et est appelée « l'attention ». On ne peut pas « faire attention » à tout ce qui nous entoure. L'attention trie donc les informations intéressantes et élimine celles sans rapport avec le sujet à mémoriser.

Cet aperçu général du système humain de traitement de l'information permet de distinguer deux choses : d'un côté existent des structures de stockage (les registres sensoriels, la mémoire à court terme, la mémoire à long terme); de l'autre existent des processus qui permettent d'agir sur les informations contenues dans ces structures (reconnaissance des modèles, répétition, attention,...). Nous allons, à présent, expliquer davantage ces structures et ces processus.

2.1.1.1. Les registres sensoriels

Nous avons mentionné plus haut le registre visuel stockant les informations du sens visuel. Il existe aussi un registre sensoriel pour les quatre autres sens : l'ouïe, le toucher, l'odorat et le goût. Les registres visuel et auditif ont été les plus valablement étudiés par les psychologues. C'est pourquoi nous en parlerons plus souvent.

Un registre sensoriel sert à stocker des informations concernant un stimulus, un court instant, sous la forme dans laquelle il a été présenté initialement au système, jusqu'à ce qu'il soit mis sous une autre forme et envoyé plus loin dans le système.

Le temps pendant lequel les informations restent dans les registres est très court car celles-ci sont sujettes à une extinction rapide. Mis à part l'extinction, les informations des registres sensoriels peuvent être enlevées de ceux-ci parce que de nouvelles informations y entrent. Si ce n'était pas le cas, les registres seraient submergés.

2.1.1.2. La reconnaissance des modèles

Elle intervient entre les registres sensoriels et la mémoire à court terme. C'est un processus de mise en correspondance des informations sensorielles avec celles qui sont en mémoire à long terme.

Son but est de convertir des informations brutes et inutiles pour le système en informations ayant du sens pour celui-ci (par exemple, donner un nom au stimulus). Ce processus est indispensable. En effet, si on catégorisait un cheval comme étant un ours, cela pourrait poser des problèmes.

Le processus de reconnaissance des modèles est un processus très complexe. Comment sommes-nous capables de reconnaître les lettres parmi toutes les formes et tailles d'écritures différentes ? Une catégorie peut, en effet, inclure un grand nombre de modèles. Par exemple, la lettre « A » peut aussi s'écrire « a », « α » ou « **A** ». De plus, ces mêmes modèles peuvent apparaître sous différentes orientations et tailles. Mais, le plus intrigant est qu'un nouveau modèle, jamais vu auparavant, peut être reconnu : « $\mathcal{A}$ », par exemple. Nous sommes, en effet, tous capables de lire une lettre écrite de la main d'un inconnu.

2.1.1.3. La mémoire à court terme

La mémoire à court terme est aussi appelée « mémoire immédiate », « mémoire de travail » ou « mémoire primaire ». Les deux premières appellations montrent bien que la mémoire à court terme contient ce sur quoi on travaille, ce à quoi on pense.

Les informations résultant du processus de reconnaissance des modèles peuvent être envoyées à la mémoire à court terme. Il y a deux limites à la mémoire à court terme. La première est une limite de capacité : seuls une demi-douzaine d'items peuvent être stockés simultanément en mémoire à court terme. La seconde est une limite de temps : s'il n'est pas répété, un item reste dans la mémoire à court terme moins de trente secondes.

La répétition a donc une grande importance : elle rafraîchit les items en mémoire à court terme, c'est-à-dire qu'elle les replace chaque fois en première position, elle les maintient en mémoire à court terme, ce qui permet de ne pas les oublier.

La répétition est aussi efficace lorsqu'elle se fait mentalement; elle est alors appelée « subvocalisation ». La répétition semble, de plus, avoir une deuxième fonction : plus un item est répété, plus il reste longtemps en mémoire à court terme; plus il est donc facile de le rappeler. Cela signifie que la répétition sert au transfert des informations de la mémoire à court terme à la mémoire à long terme, qu'elle permet d'enraciner les informations en mémoire à long terme et qu'elle facilite leur rappel.

2.1.1.4. La mémoire à long terme

La mémoire à long terme est une structure de stockage très complexe. Son étude du point de vue des matériaux verbaux (listes de mots) a conduit à d'importantes découvertes mais aussi à des limitations. En effet, mémoriser une liste de mots est différent de mémoriser une conversation ou un formule algébrique.

On peut se rendre compte de la complexité de la mémoire à long terme en réalisant qu'elle contient tout ce qu'on connaît sur le monde qui nous entoure. De plus, chaque information qu'elle contient peut être retrouvée par différents chemins d'accès. En général, les informations contenues dans la mémoire à long terme sont rangées de manière à ce que leur recherche soit relativement aisée. Par exemple, on peut retrouver un mot par sa définition ou par des mots qui lui ont été associés.

La rapidité d'accès à un mot qui se trouve en mémoire à long terme montre que celle-ci doit être extrêmement bien organisée.

Un autre point important, à propos de la mémoire à long terme, est que, contrairement aux registres sensoriels et à la mémoire à court terme, elle est un moyen de stockage permanent. Mais alors, pourquoi n'arrive-t-on pas toujours à retrouver ce que l'on veut dans notre mémoire ?

Certains pensent que l'oubli est un problème de recherche des informations en mémoire à long terme. Les informations sont disponibles, mais on arrive pas à y accéder. On ne retrouve pas le ou les chemins d'accès qui conduise(nt) à l'information recherchée.

Un dernier point intéressant à signaler est que la mémoire à long terme contient des informations de types fort différents, concernant un même item. Par exemple, pour le mot « train », la mémoire à long terme contient notamment des informations sur la façon dont il est écrit et prononcé, sur sa signification, sur l'apparence d'un train. C'est pourquoi certains théoriciens pensent que la mémoire à long terme pourrait être divisée en plusieurs parties distinctes.

2.1.1.5. L'attention.

Le concept d'attention, c'est-à-dire de la limite générale de la capacité du système à traiter les informations, a changé au cours des années de recherches. L'attention a été définie comme étant un goulot, un filtre du système. Celui-ci est nécessaire car la quantité d'information affectée aux sens à chaque instant est énorme. En effet, quand on lit un livre, on reçoit, en plus des informations relatives aux pages du livre, des informations sur les odeurs, les bruits qui nous entourent. Parmi ces informations, certaines sont importantes pour le système, d'autres pas. L'attention trie donc ces informations, garde celles qui doivent passer dans la mémoire à court terme et élimine celles qui n'ont aucune importance. C'est pour cette raison qu'elle est appelée « attention sélective ».

Le concept d'attention considéré comme un filtre a dérivé vers une autre définition, celle de capacité limitée susceptible d'intervenir à tout moment et à chaque niveau de traitement dans le système. Cette limite se retrouve avant la reconnaissance des modèles, par exemple, pour choisir les modèles qui doivent être reconnus, ou à l'entrée de la mémoire à court terme. Mais il est préférable de parler de processus d'attention car l'attention n'est pas une structure située à un endroit précis dans le système, mais bien un processus qui demande de l'attention.

Maintenant que le système de base a été expliqué, nous allons exposer une théorie qui reprend les concepts vus ci-dessus et qui spécialise le système de traitement humain des informations : « la théorie duplex » (voir Biblio 2, page 14).

2.1.2. La théorie duplex

La théorie duplex divise la mémoire en deux : la mémoire à court terme et la mémoire à long terme. La question « Existe-t-il une seule mémoire ou deux mémoires ? » reste ouverte. Freud distinguait déjà la pensée consciente (ce qui correspond à la mémoire à court terme) et la pensée inconsciente (ce qui correspond aux événements passés, donc à la mémoire à long terme). De nombreuses études et expériences tentent de prouver cette théorie mais il existe aussi beaucoup de contre-exemples. Nous allons, dans un premier temps, parler des preuves de la théorie duplex. Ensuite, nous exposerons les contre-exemples.

2.1.2.1. Les preuves de la théorie duplex

La première preuve est physiologique. Elle résulte des études de B. Milner (1959). Ayant observé un groupe de personnes atteintes à l'hippocampe (partie du cerveau nécessaire à la mémorisation), Milner constate que les patients sont capables de répéter une liste de mots immédiatement après l'avoir lue et même quelques minutes plus tard si on les autorise à répéter cette liste sans arrêt. Cependant, ils sont incapables de se souvenir des événements récemment passés ou de retenir de nouvelles informations à long terme. Cela prouve que ces personnes ont une mémoire à court terme et une mémoire à long terme intactes mais que les connections entre les deux sont détruites. Elles ne savent plus transférer de nouvelles informations en mémoire à long terme. Cette maladie est appelée syndrome de Milner.

Une autre preuve de la théorie duplex vient d'une expérience d'empan. Les personnes testées doivent lire une courte liste de lettres et la répéter immédiatement. Les erreurs qui apparaissent lors du rappel sont principalement des erreurs de confusion entre deux lettres proches l'une de l'autre du point de vue auditif (par exemple, entre B et V).

Si l'on modifie cette expérience pour l'appliquer à la mémorisation à long terme (les sujets ont une liste de mots à mémoriser et doivent la répéter une heure plus tard), les erreurs de confusion ne sont plus acoustiques mais sémantiques. Les mots sont remplacés par d'autres, de signification identique. Par conséquent, les erreurs de la mémorisation à court terme sont acoustiques tandis que les erreurs de la mémorisation à long terme sont sémantiques. Cela montre que les items sont codés par leur sonorité dans la mémoire à court terme et qu'ils sont codés par leur sens dans la mémoire à long terme.

Une dernière expérience attestant la théorie duplex est celle du rappel libre. On propose aux sujets une liste de 40 mots à mémoriser au rythme d'un mot toutes les deux secondes. Lors du rappel de ces mots, on remarque que certains sont plus facilement retenus que d'autres. La figure II.2, ci-dessous, visualise le pourcentage de mots correctement rappelés en fonction de leur position dans la liste:

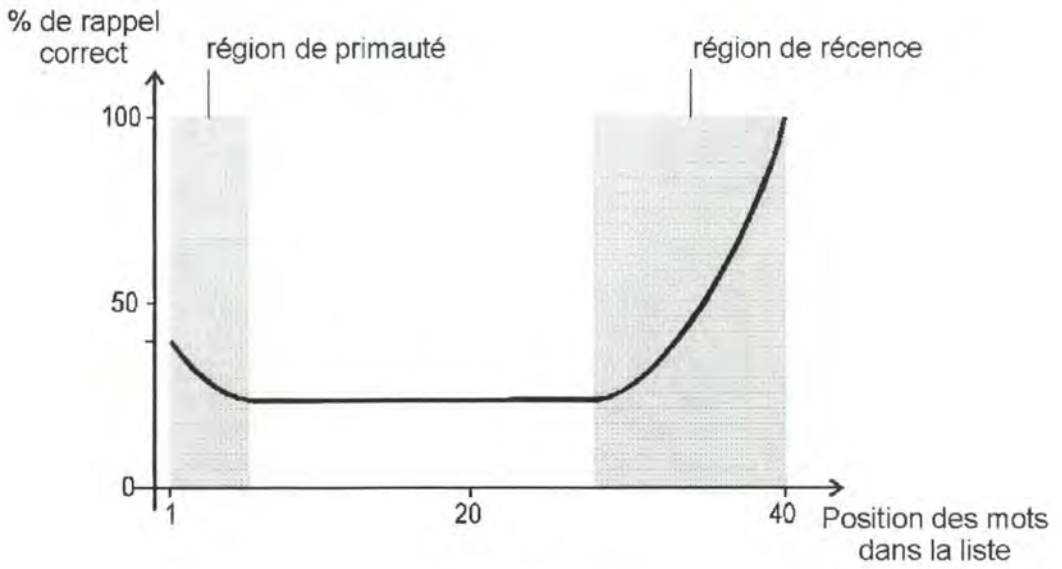


FIGURE II.2

La première portion de courbe, appelée « région de primauté », représente le rappel des mots lus en premier lieu. La dernière portion de courbe, appelée « région de récence », représente le rappel des mots lus en dernier lieu. Les mots du début et de la fin de la liste sont mieux rappelés que les autres. Les performances accrues de ces deux régions sont dues respectivement à l'effet de primauté et à l'effet de récence. La théorie duplex explique ces deux effets.

L'effet de primauté résulte du rappel des mots se trouvant dans la mémoire à long terme. En effet, les mots de la région de primauté arrivent dans une mémoire à court terme vide. Les sujets peuvent alors se concentrer sur ceux-ci et les répéter plusieurs fois avant qu'ils ne soient remplacés par les suivants. Après quelques mots (± six mots), la mémoire à court terme est remplie et les sujets n'ont le temps de répéter ces mots qu'une seule fois. Les premiers mots ont ainsi plus de chance de se trouver en mémoire à long terme que les suivants. C'est pourquoi leur taux de rappel est plus élevé.

L'effet de récence résulte du rappel des mots se trouvant dans la mémoire à court terme. En effet, les mots de la région de récence sont les derniers lus. Par conséquent, ils ne sont pas remplacés par d'autres. Ils restent donc dans la mémoire à court terme jusqu'à leur rappel.

Deux variantes à cette expérience montrent comment on peut manipuler la courbe pour affecter soit la mémorisation à court terme, soit la mémorisation à long terme.

La première postpose le rappel des mots de trente secondes pendant lesquelles les sujets doivent faire des opérations arithmétiques, ce qui les empêche de répéter les mots à rappeler. L'effet de récence disparaît alors car les derniers mots contenus dans la mémoire à court terme sont remplacés par les informations relatives aux calculs demandés. L'effet de cette manipulation est illustré, ci-dessous, sur la figure II.3 :

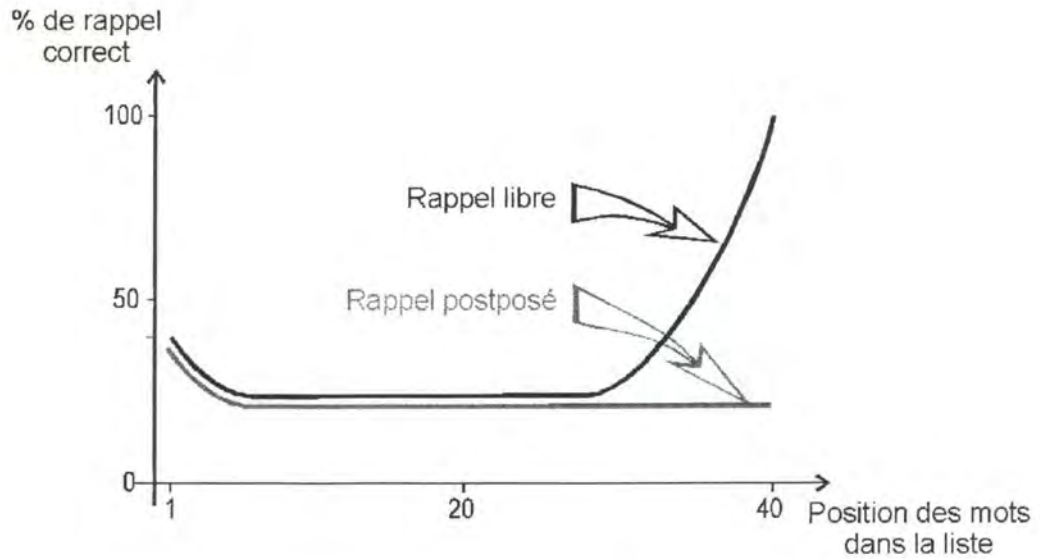


FIGURE II.3

La seconde manipulation accélère le rythme de présentation. Si celui-ci passe à un mot par seconde, les sujets sont incapables de répéter les mots plusieurs fois. De plus, les mots de la région de primauté et du milieu de la liste ont un pourcentage de rappel plus bas que lors de l'expérience de base. La mémoire à long terme stocke donc ces mots de la même manière (puisqu'ils sont répétés le même nombre de fois), ce qu'illustre la figure II.4 :

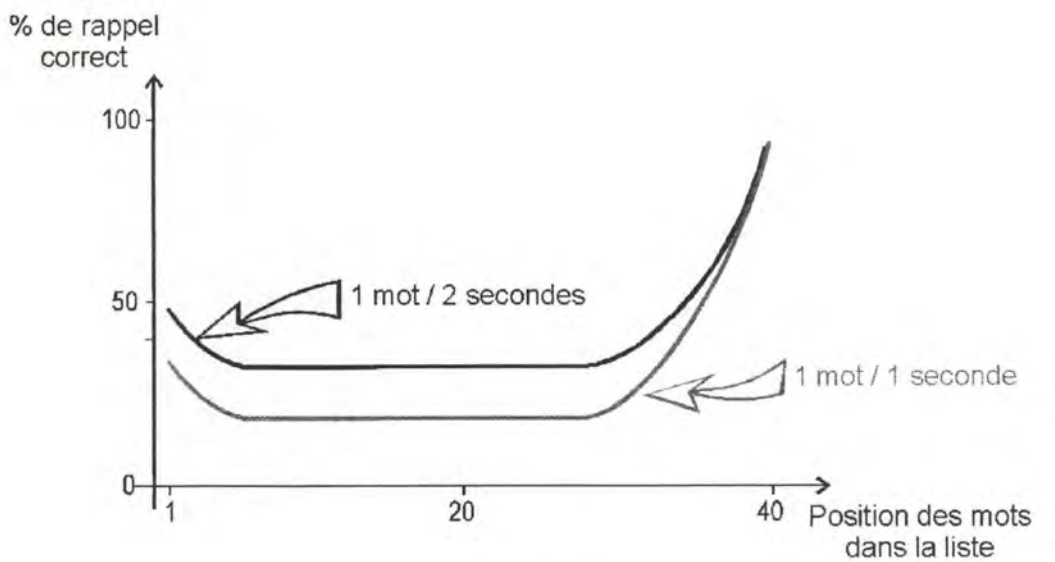


FIGURE II.4

2.1.2.2. Les contre-exemples

La théorie duplex est acceptée par un grand nombre de théoriciens mais certaines expériences et certains chercheurs tentent de prouver qu'elle n'est pas fondée. Ainsi, Wickelren (1973) a démontré que six des neuf preuves de la théorie duplex ne sont pas valables parce qu'elles pouvaient être interprétées autrement.

De leur côté, Bjork et Whitten (1974) ont découvert que le rappel postposé, accompagné d'une tâche de calcul mental, n'affectait pas toujours l'effet de récence, ce qui dément donc le fait que l'effet de récence dérive du rappel des mots de la mémoire à court terme.

L'interprétation du syndrome de Milner a aussi été attaquée : les patients utilisés pour l'expérience n'étaient pas tous affectés de la même manière, ce qui tend à rendre les résultats de cette expérience inconsistants.

La preuve donnée par les différentes formes de codages a également été détruite. Les informations en mémoire à long terme sont censées être codées par leur signification. Mais alors, comment expliquer le fait que l'on peut reconnaître des sons entendus il y a très longtemps ? Ceux-ci se trouvent donc dans la mémoire à long terme sous une autre forme (sonore).

En bref, si l'on accepte le fait qu'il existe deux mémoires séparées, il ne faut pas oublier qu'elles sont fortement interdépendantes. En effet, non seulement la répétition en mémoire à court terme conduit au transfert des informations en mémoire à long terme, mais encore, la mémoire à long terme contribue à l'encodage des informations en mémoire à court terme (grâce au processus de reconnaissance des modèles).

Pour conclure, il nous reste à ajouter que les modèles de la mémoire sont nombreux. Nous en avons choisi un comme référence, en l'occurrence le modèle des deux mémoires. Nous verrons dans les chapitres suivants comment cette théorie peut nous aider à comprendre la méthode de mémorisation utilisée par bon nombre de personnes et comment on peut l'améliorer.

2.2. Les techniques de mémorisation

De nombreuses méthodes, appelées moyens mnémotechniques, aident à la mémorisation. Une partie d'entre elles sont répertoriées et expliquées dans le livre de Françoise Gauquelin (Biblio 4), livre sur lequel nous nous sommes en partie fondés. A côté de ces techniques de mémorisation, Françoise Gauquelin donne quelques conseils à suivre pour améliorer et accroître la mémoire.

Ces derniers ont retenu notre attention :

Tout d'abord, Françoise Gauquelin prétend que « la mémorisation par répétition est plus efficace si l'on fragmente le texte à apprendre ». Ce premier conseil doit néanmoins être accompagné d'une utilisation fréquente de l'information mémorisée. En terme de liste de mots, cela signifie qu'apprendre une liste par petits groupes et la réviser souvent est plus efficace que l'apprendre en une fois ou ne jamais la réviser.

Le deuxième conseil est l'exercice de l'empan : on peut améliorer la capacité de notre mémoire à court terme en s'exerçant à mémoriser des listes de chiffres ou de mots. Nous avons vu plus haut que la capacité de notre mémoire était, en général, de 7 ± 2 éléments. Avec de l'exercice, on peut augmenter cette capacité. Nous avons donc repris cet exercice de l'empan de mémoire parmi les tests à accomplir avant de commencer la mémorisation des listes de mots.

Françoise Gauquelin propose un autre test : celui des modes de mémorisation, petit « check-up » précisant les points forts et les points faibles de notre mémoire. En fonction de ces résultats, on peut soit se concentrer sur les points forts pour mémoriser plus rapidement et plus efficacement, soit développer les points faibles pour posséder plus d'atouts dans son jeu. Dans les deux cas, la méthode de mémorisation s'en trouve améliorée.

3

PRESENTATION DU DIDACTICIEL

Par Valérie Walthéry

Le didacticiel d'aide à la mémorisation se compose de trois parties :

- le test sur les modes de mémorisation;
- le test sur l'empan de mémoire;
- l'apprentissage des listes de mots.

L'apprentissage des listes de mots (ou de nombres) est la partie essentielle de ce mémoire. Elle permet d'aider l'utilisateur dans l'apprentissage de ces listes. Elle consiste à gérer cet apprentissage en suivant l'utilisateur tout au long de celui-ci.

Cette partie centrale est encadrée par deux tests, très intéressants pour l'utilisateur, concernant l'étude des listes de mots. Les informations fournies aideront l'utilisateur à choisir judicieusement les options les plus adéquates à ses aptitudes. Il pourra ainsi améliorer ses performances et mémoriser les listes plus efficacement.

Le premier test détermine les modes de mémorisation privilégiés de l'utilisateur, c'est-à-dire par exemple, le sens le plus développé chez lui ou la facilité à retenir les idées générales ou les détails d'un texte.

Le deuxième test détermine l'empan de mémoire de l'utilisateur, c'est-à-dire le nombre de mots ou de chiffres qu'il lui est possible d'emmagasiner dans sa mémoire à court terme et qu'il est donc capable de répéter fidèlement quelques secondes après les avoir lus.

3.1. Le test sur les modes de mémorisation

Ce test est tiré du livre de Gauquelin (Développer sa mémoire : méthode Richaudeau, Biblio 4, pages 11 à 24). Il permet de déterminer les modes de mémorisation privilégiés de l'utilisateur, donnant une idée du profil psychologique de sa mémoire.

Il se compose de deux parties : la première reprend une série de questions sur les expériences passées du sujet; la deuxième explore divers secteurs de la mémoire et évalue les aptitudes naturelles du sujet à mémoriser.

Il fournit des renseignements à propos de trois domaines :

1° le mode sensoriel d'acquisition

Les informations s'acquièrent par l'intermédiaire des cinq sens. Chaque personne privilégie un ou plusieurs de ces sens. Le test détermine lequel d'entre eux est privilégié par rapport aux autres. Malheureusement, il ne donne des indications qu'en ce qui concerne le sens visuel et le sens auditif. Par conséquent, si aucun de ces deux sens n'est privilégié par rapport à l'autre, on dira que le sujet a un mode sensoriel d'acquisition synesthésique.

2° le mode de rétention mnémonique

Certaines personnes ont un esprit de synthèse, d'autres retiennent plus facilement les détails. Le test permettra de déterminer si la rétention mnémonique du sujet privilégie « l'esprit de géométrie ou l'esprit de finesse » (voir Biblio 4), c'est-à-dire s'il a la mémoire des idées ou la mémoire des chiffres.

3° l'orientation mnémonique

Elle révèle si la mémoire de l'utilisateur est plutôt tournée vers les réalités matérielles, les faits concrets, les problèmes pratiques ou si elle est plutôt tournée vers les données abstraites, immatérielles, les pensées intuitives.

Ce test est très utile en ce qui concerne l'apprentissage des listes car, en fonction des résultats obtenus, l'utilisateur pourra choisir la méthode d'apprentissage la mieux adaptée à son profil. Par exemple, si l'utilisateur a le sens visuel plus développé, il apprendra plus facilement ses listes par la méthode dite passive (voir chapitre 3.3.1.2). Dans le cas contraire, en répétant à haute voix les listes qu'il veut connaître, il se simplifiera la tâche.

Le test tel qu'il est présenté dans le livre se trouve ci-après (pages 26 à 30).

Quel est votre type de mémoire ?

1. Que faites-vous pour ne pas oublier une obligation qui risquerait de vous échapper?

☐ a) Un noeud à mon mouchoir.
☐ b) Je le note sur une feuille volante destinée à être placée bien en vue ou à portée de ma main comme un rappel constant.
☐ c) Je la note sur mon agenda.
☐ d) Rien : je n'oublie jamais rien.

2. Votre agenda est-il régulièrement tenu à jour?

☐ a) Oui.
☐ b) Assez régulièrement ; par moments j'oublie tout de même d'y noter ce que je devrais.
☐ c) J'utilise rarement un agenda ; c'est trop ennuyeux.

3. Actuellement vous semble-t-il plus facile de mémoriser?

☐ a) Des chiffres,
☐ b) Des renseignements sous forme de graphiques ou schémas,
☐ c) Des vers rythmés ou mélodies,
☐ d) Aucun de ces trois domaines ne vous est facile.

4. En classe, préfériez-vous être interrogé?

☐ a) En algèbre,
☐ b) En géométrie,
☐ c) En langues vivantes,
☐ d) En langues mortes.

5. Pour apprendre par coeur, était-il pour vous plus efficace de:

☐ a) Réécrire vous-même les données à mémoriser,
☐ b) Répéter plutôt à mi-voix ces données,
☐ c) Autre technique adoptée.

6. Résumer en une phrase sur la ligne suivante le premier souvenir d'enfance qui vous est resté:

.....
.....

A présent, classez-le dans une des catégories suivantes:

☐ a) Perception sensorielle indifférenciée,
☐ b) Impression visuelle avant tout,
☐ c) Impression auditive avant tout,
☐ d) Problème affectif,
☐ e) Problème de raisonnement.

7. Si un ami voulait vous offrir pour votre anniversaire un abonnement à une revue, choisiriez-vous qu'elle traite de préférence de:

☐ a) Littérature,
☐ b) Poésie,
☐ c) Musique,
☐ d) Peinture,
☐ e) Arts graphiques,
☐ f) Historique,
☐ g) Politique,
☐ h) Bricolage,
☐ i) Jardinage,
☐ j) Cuisine.

8. Vous souvenez-vous d'avoir rêvé?

☐ a) Souvent,
☐ b) Parfois,
☐ c) Très rarement.

9. Vous rappelez-vous avoir fait des rêves où la couleur imposait ses tonalités?

☐ a) Rarement: la couleur ne joue en général pas un rôle dans mes rêves.
☐ b) Jamais.
☐ c) Oui, souvent.

10. Des sons imaginaires interviennent-ils souvent dans vos rêves?

☐ a) Oui, les sons que je rêve peuvent même devenir si intenses que parfois ils me réveillent.
☐ b) Le matin, il m'arrive parfois de rêver que le réveil sonne, avant qu'il ne sonne en réalité. A part cela, je ne me rappelle pas avoir rêvé de sons imaginaires.
☐ c) Je n'en sais rien.

11. Des mets que vous goûtez évoquent-ils parfois nettement pour vous des couleurs ou des sons?

☐ a) Non, jamais. Leur goût me suffit.
☐ b) Oui: Une couleur peut s'associer pour moi à un goût.
☐ c) Oui: Un son peut s'associer pour moi à un goût.

12. Une odeur agréable peut-elle s'associer pour vous à une couleur ou à un son?

☐ a) A une couleur, oui: je vois une rose quand je sens l'odeur d'une rose; je vois un jambon quand je sens l'odeur du jambon.
☐ b) De façon très indistincte: oui, de telles associations se produisent chez moi sans raison apparente.
☐ c) Il m'arrive souvent très nettement d'associer des perceptions d'origine différente, je ne sais pourquoi, mais c'est un jeu qui me plaît.
☐ d) Non jamais.

13. Avez-vous fait parfois l'expérience décrite par Proust sur la madeleine trempée dans une tasse de thé: une sensation caractéristique a-t-elle réveillé en vous une succession de souvenirs que vous croyiez effacés?

☐ a) Oui, cela m'est arrivé plusieurs fois.
☐ b) Il me semble que oui, mais je n'en suis pas certain.
☐ c) Non, cela ne m'est jamais arrivé.

14. Dans vos lectures quotidiennes, qu'estimez-vous plus facile à enregistrer?

☐ a) Les idées générales traitées.
☐ b) Plutôt les données chiffrées expérimentales.
☐ c) Je n'enregistre en général rien du tout, car je lis pour m'amuser seulement.

15. Vous est-il arrivé, devant des lettres isolées, ou une poésie, ou une liste de mots qui se suivent sans ordre, de faire l'expérience décrite par Rimbaud dans son poème sur les voyelles: avez-vous vu les lettres ou les mots parés de couleurs spécifiques?

☐ a) Oui, très souvent.
☐ b) Je ne sais pas.
☐ c) Non, certainement pas.

16. Quand vous lisez un livre d'histoire, avez-vous l'impression de retenir plus facilement:

☐ a) Les dates plutôt que les noms propres,
☐ b) Les noms propres plutôt que les dates,
☐ c) Ni les dates, ni les noms, mais uniquement la succession des événements relatés.

17. Quand vous reprenez une lecture interrompue la veille, vous est-il facile de vous rappeler le chiffre de la page à laquelle vous en êtes resté?

☐ a) Non, cela ne m'arrive jamais; je préfère mettre une marque à la page où je suis arrivé.
☐ b) Je ne me souviens pas de la page exacte, mais seulement de l'endroit approximatif d'après les idées que je retrouve en feuilletant.
☐ c) Oui, je me rappelle sans peine la page exacte si j'y ai prêté garde.

18. Lorsqu'un objet vous intéresse, vous rappelez-vous son prix?

☐ a) Oui, sans difficulté.
☐ b) Non, sauf si je désire beaucoup l'objet.
☐ c) Je souhaite me rappeler le prix des choses, mais je ne peux jamais y parvenir: je les oublie à peine les ai-je lus.

19. Connaissez-vous le montant actuel de votre compte en banque (ou de vos économies en argent liquide, si vous n'avez pas encore de compte en banque)?

☐ a) Il doit se situer aux environs de... mais je ne connais pas son montant exact.
☐ b) Oui, très exactement: il est égal à...
☐ c) Je ne peux dire de mémoire où il en est; je n'ai qu'à consulter le dernier relevé pour le connaître.

20. Pendant <u>trois minutes au maximum</u>, notez tous les numéros utiles que vous savez par coeur (Exemple: votre numéro de téléphone, celui de vos amis et connaissances, votre numéro de sécurité sociale, de carte d'identité ou de compte bancaire):

.....
.....
.....

A la fin des trois minutes, faites votre total.
Nombre de numéros notés: .....

21. Pendant <u>trois minutes au maximum</u>, notez le plus possible de dates apprises en histoire, en signalant à quel événement elles se rapportent (Exemple: an 0, naissance de J.-C.):

.....
.....
.....

A la fin des trois minutes, faites votre total.
Nombre de dates notées: .....

22. Pendant <u>trois minutes au maximum</u>, notez les dates de naissance complètes (jour, mois, année) que vous savez par coeur. (Exemple: votre date de naissance, celle des membres de votre famille, de vos amis et relations, d'hommes ou de femmes célèbres qui vous intéressent):

.....
.....
.....

A la fin des trois minutes, faites votre total.
Nombre de dates de naissance complètes: .....

23. Pendant <u>cinq minutes au maximum</u>, énumérez les poésies dont vous savez par coeur une strophe au moins, en les identifiant par quelques mots du début. (Exemple: Maître Corbeau sur un arbre perché):

.....
.....
.....

A la fin des cinq minutes, faites votre total.
Nombre de poésies identifiées: .....

24. Pendant <u>cinq minutes au maximum</u>, notez de même quelques mots du début des chansons dont vous savez un couplet au moins. (Exemple: Allons Enfants de la Patrie):

.....
.....
.....

A la fin des cinq minutes, faites votre total.

Nombre de chansons identifiées: .....

25. Pendant <u>cinq minutes au maximum</u>, cherchez les noms et les prénoms de camarades de classe dont vous vous souvenez, en précisant si possible également quelle classe ils suivaient avec vous. (Exemple: 5° Philippe Souchet, Anne-Marie..., 6° Gendrot, Pascal Cossonet, Claudia...)**:**

.....
.....
.....

A la fin des cinq minutes, faites votre total: 2 points pour chaque nom et prénom associés, 1 point pour chaque nom ou prénom isolé. Dans l'exemple, on a un total de 7 points.

Nombre de noms et prénoms identifiés: ..:.....

26. Pendant <u>une minute au maximum</u>, regardez avec attention les petits dessins géométriques suivants. Puis couvrez-les avec votre main gauche et reproduisez, dans le cadre à droite, tous les dessins dont vous vous souviendrez. L'ordre de la reproduction importe peu. Ce qui compte, c'est la fidélité des dessins.

Corrigé: Comptez 1 point pour chaque reproduction exacte, ½ point pour une reproduction déformée.

Votre total: .....

27. Pendant <u>une minute au maximum</u>, étudiez les mots en désordre indiqués ci-dessous. Ensuite, couvrez-les avec votre main gauche et notez, dans le cadre à droite, tous les mots dont vous vous souviendrez, dans l'ordre où vous les retrouverez.

Corrigé: Comptez 1 point pour chaque mot exact, ½ point pour un mot déformé.

Votre total: .....

28. Pendant <u>une minute au maximum</u>, regardez l'image suivante. Puis, masquez-la, et notez dans l'espace blanc qui suit, les éléments de l'image dans l'ordre où ils vous reviennent.

Nombre d'éléments de l'image notés : .....

29. Lisez une fois pour vous-même, lentement et à haute voix, la phrase suivante. Ensuite, vous la dissimulerez et noterez dans l'espace réservé à cet effet ce que vous aurez retenu. Vous procéderez de même pour les deux phrases qui suivent la première.

Première phrase à lire, puis à reproduire de mémoire :
« J'en ai vu l'exemple en une grand-mère fort instruite qui n'arriva jamais à enseigner à sa petite fille le calcul et l'orthographe. »

Masquez la phrase et reproduisez-la ici :
.....
.....
.....

Deuxième phrase :
« Au temps des concours de récitation, celui qui n'était pas sûr de sa mémoire trichait un peu, non pas pour conquérir une bonne place, mais pour éviter une punition. »

Masquez la phrase et reproduisez-la ici :
.....
.....
.....

Troisième phrase :
« Les jeunes musiciens ressemblent assez aux physiciens de la dernière minute qui nous lancent des paradoxes sur les temps et les vitesses. »

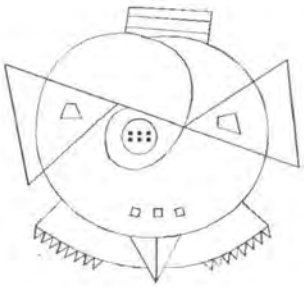
Masquez la phrase et reproduisez-la ici :
.....
.....
.....

A présent, corrigez vos réponses en cherchant les inexactitudes qu'elles contiennent :

1. Soulignez d'un trait les expressions ou les mots reproduisant l'idée, mais pas le mot à mot textuel. Exemple : « qui ne parvenait pas » à la place de « qui n'arriva jamais ».
2. Mettez un 0 lorsqu'un mot manque complètement. Exemple : « à sa petite fille 0 l'orthographe » au lieu de « à sa petite fille le calcul et l'orthographe ».

29/1. Nombre de soulignés pour l'ensemble des trois phrases : .....
29/2. Nombre d'omissions pour l'ensemble des trois phrases : .....

30. Pendant une minute au maximum, regardez avec attention le graphique ci-dessous, en observant toutes ses composantes. Ensuite masquez-le, et essayez de le reproduire avec le plus de détails possible, tout en respectant aussi ses proportions.



A présent, comparez votre dessin au modèle. Notez un point par élément reproduit à peu près, deux points par élément tout à fait exact.

Votre total : .....

ETUDE DE VOS RESULTATS AU QUESTIONNAIRE

Dans le tableau suivant, encerclez votre réponse à chaque question posée. Si, une fois ou deux, vous avez choisi une lettre qui n'est pas mentionnée dans le tableau, ne vous inquiétez pas: il s'agit de réponses normales, fréquemment données, mais qui ne sont pas significatives dans le cadre de ce test. Passez tout simplement à la ligne suivante. Puis lorsque le tableau est rempli, faites le total par colonne de vos réponses.

TYPE	I	II	III	IV	V	VI	VII
Question							
1				c		b	a
2				a			c
3		c	b		a		
4		c	b	d	a		
5		b	a				
6	a	c	b		e		d
7		e f	g h	b c		i j k	a d
8						c	a
9			c		b	a	
10		a	c			b	
11	b c					a	
12	c					a	b
13	a					c	b
14				a	b		c
15			a			c	b
16		b		c	a		
17				b	c		
18					a	b	c
19					b		c
20					10 et +	5 à 9	4 et -
21-22					10 et +	5 à 9	4 et -
23		4 et +	3 et -				
24		4 et +	3 et -				
25		7 à 14		6 et -			15 et +
26			7 et +			9 à 14	4 et -
27			15 et +	8 et -		5 et 6	
28			10 et +	5 à 9			4 et -
29/1 29/2		0 à 5		5 et +			
30			15 et +			5 à 14	4 et -
Totaux							

INTERPRETATION

Les *trois premières colonnes* du tableau étudient quel mode sensoriel est le plus favorable chez vous aux acquisitions mnémoniques.

La colonne I correspond à des *acquisitions synesthésiques*, c'est-à-dire, où tous vos sens participent, sans qu'aucun ne soit privilégié. C'est votre cas si vous avez encerclé *trois éléments ou plus* dans la colonne I.

La colonne II correspond à des acquisitions par le *sens auditif* avant tout. C'est votre cas si vous avez encerclé *quatre éléments ou plus* dans la colonne II.

La colonne III correspond à des acquisitions par le *sens visuel* avant tout. C'est votre cas si vous avez encerclé *cinq éléments ou plus* dans la colonne III.

Les *deux colonnes suivantes* révèlent si votre rétention mnémonique privilégie, selon l'expression de Pascal, « l'esprit de géométrie » ou « l'esprit de finesse ».

La colonne IV correspond à une mémoire qui retient surtout les *idées générales*. Vous avez cette forme de mémoire si vous avez encerclé *cinq éléments ou plus* dans la colonne IV.

La colonne V correspond à la mémoire qui retient le plus facilement les *chiffres*. Vous avez cette forme de mémoire si vous avez encerclé *quatre éléments ou plus* dans la colonne V.

Enfin, les *deux dernières colonnes* révèlent si votre type de mémoire est plutôt tourné vers les réalités matérielles ou non.

La colonne VI correspond à une capacité plus grande de retenir les *faits concrets* que les données abstraites, immatérielles. C'est votre orientation habituelle si vous avez encerclé *six éléments ou plus* dans la colonne VI.

La colonne VII correspond le plus souvent à une *pensée intuitive*. C'est votre façon de mémoriser si vous avez encerclé *six éléments ou plus* dans la colonne VII.

VOTRE PROFIL MNEMONIQUE

Vous pouvez résumer dans le tableau suivant, selon les trois axes étudiés, vos modes de mémorisation privilégiés. Pour cela, encerclez à chaque ligne le chiffre romain pour lequel le nombre de vos réponses dépasse la moyenne indiquée dans l'interprétation des résultats.

Les trois axes étudiés	Encerclez votre réponse la meilleure à chaque ligne		
Mode sensoriel d'acquisition privilégié	I synesthésique	II auditif	III visuel
Mode de rétention	IV mémoire des idées		V mémoire des chiffres
Orientation mnémonique vers	VI les faits concrets		VII l'imagination intuitive

Intérêt de l'ordinateur

Le test présenté ci-dessus comporte quelques inconvénients qui disparaissent lorsqu'on le pratique sur ordinateur :

- pour faire le test dans le livre, il faut se munir d'un chronomètre. Puisque le sujet doit garder les yeux sur celui-ci, il lui est difficile de bien se concentrer sur les questions posées. Avec l'ordinateur, l'utilisateur ne doit plus penser au temps; quand le temps est écoulé, on le lui signale;

- à la fin du test, le sujet devait reporter ses réponses dans un tableau à sept colonnes (tableau-type). Il devait ensuite totaliser les points de chaque colonne et reporter ces totaux dans un deuxième tableau qui lui indiquait, enfin, son mode sensoriel d'acquisition, son mode de rétention mnémonique et son orientation mnémonique. L'ordinateur fait tout ce travail à la place de l'utilisateur et lui donne les résultats sur simple demande.

- l'utilisateur ne pouvait faire le test qu'une seule fois dans le livre. Il n'est, en effet, pas très pratique d'effacer toutes les réponses pour pouvoir recommencer le test. Avec l'ordinateur, il peut le recommencer autant de fois qu'il le souhaite. Il pourra ainsi découvrir si son profil s'est modifié au long des exercices.

3.2. Le test sur l'empan de mémoire

Ce test est également tiré du livre de Gauquelin (Développer sa mémoire : méthode Richaudeau, Biblio 4, pages 73 à 76). Il se compose de deux tests distincts : l'empan de mémoire des mots et l'empan de mémoire des chiffres.

Il permet de déterminer l'empan de mémoire d'une personne, c'est-à-dire sa capacité moyenne à absorber des informations en mémoire à court terme; il s'agit donc du nombre de mots (pour l'empan des mots) ou du nombre de chiffres (pour l'empan des chiffres) qu'une personne est capable de stocker dans sa mémoire à court terme.

Nous avons vu plus haut que l'empan moyen de mémoire d'une personne était de 7 ± 2 éléments (voir chapitre 2.1.1). Cette approximation est valable lorsqu'il s'agit de séries de chiffres ou de mots non significatives. En effet, si l'on considère des séries de mots sensés (donc des phrases), l'empan de mémoire des mots sera plus élevé. Il faudra alors, pour retrouver la moyenne de 7 ± 2 éléments, considérer l'empan de mémoire des phrases et non plus des mots. Le but à atteindre pour améliorer l'empan sera de se construire une histoire qui associera les mots à retenir, s'imaginer un film dans lequel les mots jouent.

De son côté, l'empan de mémoire des chiffres est améliorable si l'on regroupe les chiffres à retenir par deux, trois, etc. On passera alors à l'empan de mémoire des nombres.

Ce test est très intéressant car, en fonction des résultats, l'utilisateur pourra, par exemple, décider du nombre de mots présentés par écran pendant l'apprentissage des listes. Si l'empan des mots est faible, l'utilisateur pourra diminuer ce nombre; si l'empan des mots est élevé, il pourra augmenter ce nombre.

Ce test sert aussi d'exercice si l'utilisateur décide de s'y reprendre plusieurs fois. En effet, l'empan de mémoire n'est pas figé; il s'améliore avec l'entraînement. Une étude de Martin et Fernberger, d'où est tiré le tableau ci-dessous, l'illustre :

Nombre de jours d'exercice	Empan du sujet A	Empan du sujet B
1 à 5	9	10
6 à 10	9	11
11 à 15	9	11
16 à 20	10	12
21 à 25	10	11
26 à 30	13	12
31 à 35	12	12
36 à 40	12	12
41 à 45	13	14
46 à 50	13	15

Le test tel qu'il est présenté dans le livre se trouve ci-après (pages 34 à 35).

Présentation du test de l'empan dans le logiciel

Lorsque l'utilisateur débute le test de l'empan, il se voit proposer une série de chiffres générés aléatoirement (s'il s'agit de l'empan des nombres) ou une série de mots choisis aléatoirement dans un fichier de mots (s'il s'agit de l'empan des mots). Les chiffres ou les mots sont affichés un à un à l'écran à un rythme régulier (voir annexes, écran 4). A la fin de chaque série, l'utilisateur est invité à recopier celle-ci au clavier (voir annexes, écran 6). Dès que l'utilisateur commet une erreur, l'exercice est interrompu et les résultats du test sont affichés à l'écran (voir annexes, écran 5). L'utilisateur peut recommencer ce test autant de fois qu'il le désire. Il a également la possibilité de consulter les résultats des tests précédents.

Test de votre empan de mémoire de chiffres

Prenez une feuille de papier en guise de cache, pour couvrir la colonne des nombres sur laquelle vous allez travailler, et une autre feuille de papier que vous placerez à côté de cet ouvrage pour y reporter vos résultats. Munissez-vous aussi d'un stylo pour noter ces résultats.
Servez-vous du cache pour masquer la première colonne de nombres. Puis faites apparaître le premier nombre de la colonne masquée: 1708. Prononcez chaque chiffre de ce nombre. Puis, sans regarder, notez ce nombre tel que vous l'avez retenu sur la feuille préparée à côté du livre. Faites de même pour tous les nombres suivants. Quand vous avez fini la première colonne, faites-en autant pour les deux autres colonnes.

1708	7932	5106
2467	3326	8045
9774	2467	6242
12568	85992	26969
96682	73105	03729
55595	63564	38548
172277	943949	544354
844217	533157	245506
630163	785916	955557
3321123	4297864	5607825
5780863	2440947	2796544
1818079	2464417	1658097
26623897	75841607	44998311
23424064	74829777	81074532
52362819	95509226	11970056
378594351	283395008	304234079
702917121	340332038	261389510
566218373	596835087	759712259
9949572277	8842954572	1664361600
1608150472	3327143409	4559346849
1272073445	3116933243	5027898719

Corrigé du test

Comparez à présent les nombres inscrits sur votre feuille avec ceux de la liste d'origine. Indiquez sur votre feuille les divergences, de préférence avec une couleur différente.
Notez, dans les carrés blancs ci-dessous, à partir de quelle quantité de chiffres ces divergences apparaissent pour chaque colonne.

- Dans la première colonne, la première divergence apparaît dans un nombre de ..... chiffres.
- Dans la deuxième colonne, la première divergence apparaît dans un nombre de ..... chiffres.
- Dans la troisième colonne, la première divergence apparaît dans un nombre de ..... chiffres.

Il est peu problable qu'en l'espace de trois courts exercices seulement vos performances aient eu le temps de s'améliorer. Ce que nous allons tirer de cet exercice, c'est votre empan de mémoire de chiffres actuel.
Pour le connaître, faites la moyenne de vos résultats sur les trois colonnes. Par exemple, si votre première erreur est apparue dans le nombre de 6 chiffres dans la première colonne, de 8 chiffres dans la deuxième colonne et de 7 chiffres dans la troisième colonne, additionnez 4 + 6 + 5 = 15, et divisez ce total par 3. Dans cet exemple, votre empan serait de 5 chiffres en moyenne. Ce qui veut dire que vous savez retenir en mémoire immédiate des nombres de 5 chiffres, mais avec des risques d'erreurs.
Les numéros de téléphone comprennent 6 chiffres. Il serait profitable que votre empan de mémoire des chiffres soit consolidé pour les nombres de cette dimension. Nous vous conseillons, si vos erreurs apparaissent à partir des nombres de 5 chiffres, d'exercer régulièrement votre mémorisation immédiate de chiffres.

Test de votre empan de mémoire des mots

Procédez pour cet exercice comme pour le précédent: placez sous la première ligne de la feuille qui vous sert de cache, prononcez chaque mot de cette ligne, puis sans les regarder à nouveau, essayez de reproduire ces mots sur une feuille à part. Faites ensuite de même pour la deuxième ligne, et pour toutes les lignes suivantes.
Le chiffre en retrait indique le nombre de mots par ligne.

3. réponse - châtaigner - financier

4. cygne - reproduction - fenêtre - plutôt
   temps - combat - extension - beauté
   trompette - caillouteux - complication - vigoureux

5. beuverie - caréner - obscure - entrepont - attrayant
   conséquence - expédition - vitre - puni - proéminent
   poitrine - doucement - bassin - réveiller - ingrat

6. grue - ensuite - oeuf - voyage - appliqué - ramper
   brunir - livre - précipice - souverain - mangé - sinistre
   descendre - taxi - baron - alcool - inégalité - province

7. bienfaiteur - oublier - léthargie - cannelé - tour - assistance - obéissance
   cordialité - plongeon - prolonger - marteau - nettement - immérité - mauvais
   comté - dindon - cela - valve - avant-poste - élargir - isolément

3. solennité - guetter - prévoyant

4. Breton - latitude - tâche - café
   excessif - chauffeur - compétence - certain
   longe - reculer - question - exposant

5. prose - ressource - intermittent - roux - Corse
   habiter - perroquet - nicher - raisin - paquebot
   communiste - Canada - débauche - engloutir - estimation

6. mirage - boucle - référendum - douairière - absolu - dominant
   aqueux - lunatique - problème - aptitude - avec - laser
   couture - thé - réaliser - plupart - ensemble - maison

7. étaient - vouloir - concert - posté - démarche - foyer - reins
   femme - apporté - spécialement - criant - beaucoup - dit - gâteau
   amour - ceci - école - réflexion - auparavant - volonté - carré

3. objection - sont - couvert

4. sembler - famille - substance - dîner
   pluie - dedans - noire - océan
   passionnément - après - rapidité - dessous

5. tenir - garçon - apparence - réfléchir - simultanément
   répétitif - local - frapper - entrer - parallèlement
   quelque - derrière - féminin - repas - choeur

6. journal - profond - soleil - gentil - dortoir - apprécier
   chocolat - estimer - ouvrage - souhaiter - enseignement - ici
   mur - préparé - valse - savoir - agitation - concernant

7. chien - lorsque - vous - arriver - traverser - appétit - jeu
   depuis - vivre - scolaire - sauter - voulait - secours - appel
   correctement - piano - erreur - chaud - incandescence - fillette - alla

Corrigé de l'exercice.

Avec un stylo de couleur différente, notez ici aussi les différences entre votre liste de groupes de mots et la liste d'origine. A partir de quel nombre de mots (indiqués en petits caractères dans la marge) votre version commence-t-elle à différer du modèle:

1. dans la première liste ? ☐
2. dans la deuxième liste ? ☐
3. dans la troisième liste ? ☐

Faites la moyenne de vos trois résultats, comme précédemment. Vous saurez ainsi à partir de combien de mots votre empan de mémoire devient moins sûr.
Si vous avez réussi à reproduire une moyenne de 6 mots sans lacune, votre empan des mots est très satisfaisant. Avec une moyenne de 5 mots sans lacune, vous vous situez dans la moyenne générale, comme le montrent les travaux de A. Miller. Mais si votre moyenne se situe au-dessous de 5 mots, votre mémoire immédiate des mots n'est pas très développée. Vous avez intérêt à l'exercer pour l'améliorer.

Intérêt de l'ordinateur

Le test, tel qu'il est présenté ci-dessus, comporte plusieurs inconvénients que l'utilisation de l'ordinateur permet de résoudre :

- pour faire ce test dans le livre, il faut se munir de deux feuilles de papier : l'une sert à cacher les séries de mots ou de chiffres, l'autre sert à noter les réponses. Avec l'ordinateur, plus besoin de papier; l'utilisateur ne voit qu'une série de mots ou de chiffres à la fois et dès qu'elle disparaît de l'écran, il la recopie au clavier;

- l'utilisateur n'a plus besoin de comparer ses séries avec celles qui lui ont été présentées. L'ordinateur le fait automatiquement chaque fois qu'une série est encodée au clavier et s'arrête dès que l'utilisateur fait une faute;

- de plus, l'ordinateur présente des séries de mots ou de chiffres de façon aléatoire. Par conséquent, si l'utilisateur veut refaire le test, on ne lui présentera pas deux fois les mêmes séries;

- enfin, pour que le test soit correctement exécuté, le sujet doit lire un chiffre toutes les secondes. Sans montre ou sans chronomètre, il est difficile de respecter ce temps. L'utilisation de l'ordinateur élimine cette contrainte. En effet, l'ordinateur affiche lui-même un chiffre par seconde. Ainsi, l'utilisateur pourra se concentrer entièrement sur la mémorisation.

3.3. L'apprentissage des listes de mots

Cette partie du didacticiel est le point central de l'aide à la mémorisation. Elle propose à l'utilisateur plusieurs méthodes à suivre qui l'aideront à mémoriser les listes de mots choisies. Les deux tests effectués, nous proposons à l'utilisateur, pour l'apprentissage des listes, une paramétrisation, qui tiendra compte de quelques-unes de ses aptitudes à mémoriser. Après avoir essayé de mémoriser quelques listes, l'utilisateur se rendra peut-être compte que les paramètres choisis ne lui conviennent pas et pourra alors les modifier pour rendre son apprentissage plus efficace. Il aura, notamment, le loisir de choisir le nombre de mots présentés par écran, le nombre de bonnes réponses à donner pour que les mots étudiés soient considérés comme connus (pour un apprentissage approfondi ou non), ou encore le type d'apprentissage qu'il souhaite exécuter.

Nous allons commencer par une description globale des concepts développés dans ce didacticiel. Nous poursuivrons par une description plus détaillée des méthodes d'apprentissage (algorithmes d'apprentissage).

3.3.1. Description globale des concepts du didacticiel

3.3.1.1. Les listes

Les listes envisagées dans l'introduction comportent deux niveaux. Ce sont des listes d'items dont les items sont des listes linéaires de mots ou de nombres. Les listes linéaires sont aussi appelées sous-listes. Une liste à deux niveaux est donc constituée de plusieurs sous-listes.

Une sous-liste comporte au maximum dix mots (ou nombres). Les mots appartenant à la même sous-liste ont, normalement, un rapport entre eux. Cette association est d'ailleurs très importante car elle facilite la mémorisation des listes (voir 3.3.1.5).

Si on reprend l'exemple de l'introduction :

Rôti	Steaks	
Haricots	Salade	Tomates
Eau	Vin	

on remarque que l'on peut ajouter, au début de chaque sous-liste, un mot qui sert de référence aux autres mots :

Viande :	Rôti	Steaks	
Légumes :	Haricots	Salade	Tomates
Boissons :	Eau	Vin	

Ces premiers mots de chaque sous-liste expriment une caractéristique commune aux autres mots. Ils peuvent aussi être la synthèse des mots de la sous-liste ou être les mots par lesquels on accède aux autres mots. Ces mots sont appelés « mots-maîtres » car ils sont « maîtres » par rapport aux autres.

Le mot « Viande » sert donc de référence aux mots « Steaks » et « Rôti »; en effet, « Steaks » et « Rôti » sont tous deux des viandes. Lorsqu'une liste contient des mots-maîtres, ceux-ci sont toujours placés au début de chaque sous-liste. Il n'y a évidemment qu'un seul mot-maître par sous-liste.

Certaines listes ne possèdent pas de mot-maître lorsque, notamment, tous les mots des sous-listes ont la même importance, lorsqu'aucun mot ne peut être la synthèse des autres. Si l'on prend, par exemple, une liste de synonymes, on ne peut arbitrairement privilégier aucun des mots.

Les listes à deux niveaux reprennent les listes ci-dessous :

- les listes à une colonne : par exemple, un itinéraire. Dans la définition donnée ci-dessus, nous pourrions parler de « listes d'items dont les items sont des mots ». Ces listes n'ont pas de mot-maître (les sous-listes ne comportent, en effet, qu'un seul mot qui ne peut pas être « maître » de lui-même).

- les listes fixes (dont le nombre de mots par sous-liste est fixe) : par exemple, une liste de noms, prénoms et adresses. On peut leur donner un mot-maître (ici, cela pourrait être les noms).

- les listes variables (dont le nombre de mots par sous-liste est variable) : par exemple, une liste de vocabulaire français dont chaque mot aurait plusieurs traductions. On peut aussi leur donner un mot-maître (ici, par exemple, les mots français).

3.3.1.2. L'apprentissage des listes

Pour mémoriser une liste de mots, il faut, tout d'abord, bien la percevoir, c'est-à-dire la lire convenablement. Il faut ensuite associer les mots de la liste entre eux afin de les retrouver plus facilement dans la mémoire. Enfin, il faut répéter cette liste de temps en temps pour ne pas l'oublier.

Les différentes méthodes que nous proposons optent pour ce principe. Elles sont au nombre de trois et peuvent être regroupées comme suit :

NIVEAU 1 :	MODE 1 :	Apprentissage dit passif
	MODE 2 :	Apprentissage dit actif
NIVEAU 2 :		Révision

Le principe général de chaque méthode est décrit ci-dessous. Les avantages de ces différentes méthodes suivront :

1° niveau : l'apprentissage

1° mode : l'apprentissage dit passif

- Principe: l'utilisateur se voit présenter une ou plusieurs sous-listes qu'il doit ensuite répéter. Après quelques répétitions correctes, les sous-listes sont considérées comme connues, c'est-à-dire comme se trouvant dans la mémoire à long terme. La particularité de l'apprentissage passif est que, pour trouver les mots adéquats, l'utilisateur a sous les yeux des propositions de mots. Il doit donc choisir ceux qu'il présume corrects parmi les propositions (voir annexes, écrans 14 et 15) .

♦ Avantages:

- les différents mots proposés à l'utilisateur l'aideront à retrouver ceux qu'il cherche. Dans un premier temps, nous conseillons vivement aux utilisateurs de commencer par cet apprentissage avant de passer à l'apprentissage « actif ». Ceci permet de dégrossir le travail ultérieur.

- l'utilisateur ne sera pas obligé de réécrire (au clavier) les mots qui lui sont demandés puisqu'il lui suffira de choisir parmi ceux qui lui sont proposés. L'orthographe de ces mots ne sera pas pris en compte, ce qui sert uniquement lorsque l'utilisateur désire réécrire la liste sans fautes.

- ce type d'apprentissage permettra à l'utilisateur de gagner du temps puisqu'il ne devra pas réencoder les mots au clavier. L'ordinateur apporte quelque chose de nouveau en ce qui concerne l'apprentissage des listes par la méthode passive. Il est, en effet, impossible de procéder de la même manière avec une feuille de papier.

- ce type d'apprentissage sera aussi très utile pour les utilisateurs n'ayant pas le sens moteur développé, à savoir, ceux pour lesquels réécrire les mots n'aidera en rien à la mémorisation.

2° mode : l'apprentissage dit actif

♦ Principe: l'utilisateur se voit, également, présenter une ou plusieurs sous-listes qu'il doit ensuite répéter. Cependant, contrairement au type d'apprentissage passif, aucun mot ne sera plus proposé à l'utilisateur. Il devra donc retrouver les mots corrects dans sa mémoire, sans aucune aide extérieure, et les réécrire au clavier (voir annexes, écrans 14 et 16).

♦ Avantages:

- puisqu'il doit réécrire les mots qui lui sont demandés, l'utilisateur devra aussi connaître leur orthographe. Ce type d'apprentissage est donc plus approfondi que l'apprentissage passif. Les mots des listes seront donc connus dans le détail.

- pour les utilisateurs qui ont le sens moteur développé, la réécriture des mots peut aider à la mémorisation. Même pour ceux qui ont le sens moteur très peu développé, réécrire les mots n'est jamais un travail inutile. Le seul inconvénient est que l'utilisateur perd beaucoup de temps à tout recopier.

2° niveau : la révision

♦ Principe: les sous-listes considérées comme connues (se trouvant donc dans la mémoire à long terme) peuvent n'y rester qu'un bref instant si elles ne sont pas répétées de temps à autre. La révision a pour but de déterminer quelles sous-listes se trouvent toujours dans la mémoire à long terme de l'utilisateur en lui demandant de les répéter encore une fois. L'utilisateur aura le choix entre deux révisions: la première, passive, où il pourra choisir parmi des propositions de mots; la deuxième, active, où il devra réécrire les mots.

♦ Avantage : la révision sert principalement à ancrer les sous-listes dans la mémoire à long terme, mais aussi à déterminer celles qui ont été oubliées, dans le but de les réapprendre lors d'un prochain apprentissage.

Par Philippe Vignaux

3.3.1.3. Les différents types de listes

Dans le domaine des listes, la nature des informations à étudier est extrêmement variée. Pour nous en convaincre, il suffit de réfléchir à divers exemples de ce que nous sommes parfois amener à mémoriser : listes de courses, vocabulaire de langues étrangères, itinéraire routier, listes de synonymes et d'antonymes.

Toutes ces informations ont une structure propre. Les listes dans lesquelles nous rangeons ces informations doivent tenir compte de cette structure. Nous avons donc dégagé différents types de listes dont la mémorisation présente de l'intérêt.

Le premier type est constitué par les listes dont les sous-listes sont formées de deux items. Ce format de liste est très important vu la fréquence de son utilisation. En effet, il est par exemple destiné à l'étude du vocabulaire d'une langue étrangère, ce qui est utile pour les étudiants (voir annexes, écran 8).

Un autre type de liste intéressant permet de traiter des sous-listes constituées d'un nombre fixe d'items supérieur à deux (voir annexes, écran 9). Ces listes seront utilisées pour étudier des informations très structurées, par exemple d'ordre géographique (seraient repris dans ces sous-listes des items précisant respectivement les noms, capitales, chefs d'états, et monnaies de chaque pays). Des sous-listes de dix items maximum nous paraissent suffisantes. En effet, bien que ce type de liste soit tout à fait digne d'intérêt et satisfasse de nombreux apprentissages, nous estimons que, dans la réalité, nous rencontrons rarement des sous-listes dont la taille dépasse cinq ou six items.

Le troisième type est une variante du précédent. Il s'agit des listes dont les sous-listes ne possèdent pas toutes le même nombre d'items. C'est la situation rencontrée lors de l'étude, par exemple, d'une liste de vocabulaire français où chacun des mots possède un nombre variable de synonymes, ou d'une liste de pays suivie des différentes langues qui y sont parlées. Ici aussi, le nombre maximum d'items par sous-listes sera de dix.

Enfin, un dernier type de liste, celui qui permet d'étudier une liste dont chacune des sous-listes contient un seul item, présente de l'intérêt. Cette situation se rencontre lorsque l'on doit, par exemple, mémoriser une liste de noms, une liste de courses, un itinéraire routier. Dans ce type de liste, nous distinguons deux cas, selon que l'ordre des sous-listes importe ou non. Par exemple, dans le cas de l'étude d'un itinéraire routier, l'ordre des sous-listes est capital. Néanmoins, ce type de liste se rencontrera moins fréquemment.

Il nous faudra ainsi prévoir différents algorithmes d'apprentissage en fonction des différents types de listes que nous venons de mentionner. Il est clair que l'on n'étudie pas une liste de couples de la même façon que l'on étudie des sous-listes de cinq items. Les différents types de listes mis en évidence permettront à l'utilisateur de traiter une grande variété de listes.

3.3.1.4. Le sens d'apprentissage des listes

En fonction des types de listes (voir 3.3.1.3), nous avons défini différentes manières d'étudier celles-ci. Pour chaque type de liste, il existe en effet différents sens d'apprentissage. Lors d'un apprentissage, l'utilisateur ne souhaite pas toujours connaître tous les items de la sous-liste. Certains servent, en effet, d'items de base pour le rappel des autres. Nous désignons ces items par le terme « mot-maître » (voir définition pages 38 et 39).

Signalons que, dans la représentation schématique, une ligne (_____) représente un item d'une sous-liste et que plusieurs lignes placées sur une même ligne horizontale représentent les items d'une même sous-liste. Une représentation schématique de chaque type de liste accompagne les descriptions de celles-ci.

1) Sous-listes de deux items

a) représentation schématique

_____ _____
_____ _____
_____ _____
_____ _____
_____ _____

b) sens d'apprentissage

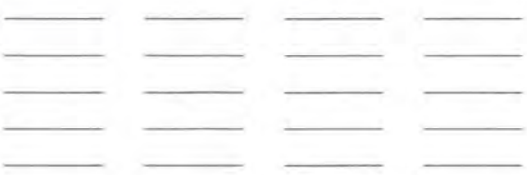
Trois sens d'apprentissage sont proposés :

- l'utilisateur désigne le premier item comme mot-maître et étudie uniquement le second. L'étude d'une liste vocabulaire français - anglais où seul le terme anglais doit être retenu est typique de cette situation.
- l'utilisateur désigne le second item comme mot-maître et étudie exclusivement le premier mot.
- l'utilisateur désigne à la fois le premier et le second item comme mots-maîtres. Il étudie alors alternativement le premier et le second item. Pour revenir à l'exemple

de l'étude d'une liste de vocabulaire français - anglais, chaque couple d'items serait étudié dans les deux sens.

2) Sous-listes de plus de deux items (taille fixe)

a) représentation schématique



b) sens d'apprentissage

Dans la représentation graphique, nous avons choisi d'illustrer, à titre d'exemple, une liste constituée de sous-listes dont la taille est toujours de quatre items. Dans ce cas, deux sens d'apprentissage sont possibles :

- l'utilisateur désigne, au choix, l'un des quatre items d'une sous-liste comme mot-maître. Ce choix est évidemment appliqué à l'ensemble des sous-listes de cette liste. Les trois autres items doivent être restitués lors de l'apprentissage.
- l'utilisateur ne désigne aucun mot-maître. Il doit alors restituer la totalité des items de chaque sous-liste lors de l'apprentissage.

3) Sous-listes de plus de deux items (taille variable)

a) représentation schématique

b) sens d'apprentissage

Deux sens d'apprentissage sont disponibles :

- le mot-maître désigné correspond obligatoirement au premier item de chaque sous-liste. L'exemple de l'introduction illustre cette situation :

  Viande : steaks - rôti
  Légumes : haricots - salade - tomates
  Boissons : eau - vin

- l'utilisateur ne désigne aucun mot-maître. Par conséquent, il doit restituer la totalité des items de chaque sous-liste lors de l'apprentissage.

4) Sous-listes d'un seul item

a) représentation schématique

_____
_____
_____
_____
_____

b) sens d'apprentissage

Un seul sens d'apprentissage est disponible. Tous les items de la liste doivent être étudiés.

3.3.1.5. Les mécanismes d'organisation

Le stock d'informations de la mémoire à long terme est immense et requiert des mécanismes d'organisation. Ainsi, dans la vie de tous les jours, de nombreux procédés mnémotechniques servent à récupérer les informations disséminées dans notre mémoire. Certains sont basés sur l'image, comme la méthode des lieux (ou *loci*) transformant les mots d'une liste en images et disposant celles-ci à chaque emplacement d'un itinéraire connu. D'autres sont des plans de récupération basés sur le langage, comme le procédé de la phrase-clé. Ces mécanismes d'organisation ont fait l'objet de nombreuses études.

Dans un premier temps sont décrits les deux grands modes d'organisation mis en évidence par les chercheurs. Par la suite sont expliqués les modes d'utilisation et d'adaptation de ces deux types d'organisation dans le programme de notre mémoire.

1) La catégorisation

Ce principe concerne, notamment, la mémorisation de listes de mots provenant de catégories conceptuelles usuelles telles, par exemple, les fleurs, les oiseaux, les métiers. Dans ce cas, les sujets auront tendance à reconstituer les catégories spontanément au rappel de la liste : ce phénomène est appelé « catégorisation ».

La catégorisation apparaît comme une possibilité d'organisation; elle donne lieu à de nombreux travaux. Ainsi, lors d'une expérience, Wood (Biblio 3) présente à un groupe de sujets cinquante-quatre mots à étudier, groupés en dix-huit catégories de trois mots. Un autre groupe apprend la même liste mais, cette fois, les mots sont mélangés. Le tableau suivant, où l'effet de catégorisation est tout à fait visible, présente les résultats aux différents rappels :

Rappel	1	2	3
Mots groupés	**17**	**28**	**39**
Mots mélangés	**11**	**20**	**29**

D'autre part, Wood remarque que, dans le groupe où les mots sont mélangés, le résultat du premier rappel, soit onze, est supérieur au nombre 7 ± 2 qui représente la quantité théorique maximale d'items que l'on peut faire entrer, en une fois, en mémoire immédiate. Cela s'explique par le fait que les sujets rattachent automatiquement les mots aux catégories conceptuelles auxquelles ils appartiennent même si, comme c'est le cas ici, les mots sont mélangés dans la liste.

a) les indices catégoriels

Dans une expérience, le chercheur Endel Tulving (Biblio 3) introduit la notion d'indices catégoriels. La tâche des sujets consiste à mémoriser une liste de mots appartenant explicitement à des catégories sémantiques bien distinctes. Pendant la phase de mémorisation, les mots sont présentés avec le nom de leur catégorie mais seuls les mots sont à rappeler (cfr. tableau ci-dessous).

Indices catégoriels	item	item	item
animal	vache	chien	pie
légume	salade	haricot	
fruit	poire		

Pour une moitié des sujets, le rappel est libre; l'autre moitié dispose, sur la feuille de réponse, du nom de chaque catégorie comme indice de récupération. L'expérience montre que le groupe qui dispose des indices catégoriels lors de la restitution a un rendement de 55% supérieur à celui de l'autre groupe. La différence de rendement s'explique essentiellement par le fait que la présentation des indices n'omet aucune catégorie. Les personnes disposant des indices de catégories restituent au total un plus grand nombre de mots. Par contre, dès qu'une catégorie est évoquée, que ce soit par présentation de l'indice ou par remémorisation du sujet, le nombre d'instances par catégories est le même.

Tulving en déduit que la présentation d'indices catégoriels rend un plus grand nombre de catégories accessible par rapport au rappel libre, tandis que ces indices restent inopérants à l'intérieur des catégories.

A la suite de Tulving, de nombreuses recherches ont été menées; elles ont permis de découvrir que les indices sont parfois de nature variée. Citons, à titre d'exemple, les indices associatifs que nous traiterons par la suite ("chaud" fait penser à "froid" et inversement) ou les indices phonétiques (la première syllabe, la rime). Ces recherches aboutirent également à la conclusion que la mémoire à court terme conservait trois ou quatre indices au maximum, faute de place pour les mots à l'intérieur des catégories. La quantité optimale d'articles par catégories fut estimée à quatre.

b) plan de récupération

Le rappel est limité par le nombre d'indices stockés en mémoire à court terme. C'est pourquoi, plus les indices sont organisés entre eux, plus le rappel est maximisé. C'est la notion de plan de récupération ou plan de rappel. Gordon Bower et ses collègues ont étudié l'efficacité d'un plan de rappel hiérarchisé en imaginant, selon une hiérarchie de catégories, une liste organisée et impressionnante de 112 mots présentée sous la forme de quatre planches d'une quarantaine de mots emboîtés dans des catégories de niveau croissant (les animaux, les plantes, les minéraux, voir Biblio 3, p.140).

Le tableau ci-dessous décrit les résultats d'un groupe ayant étudié la liste hiérarchisée par rapport à un autre groupe ayant étudié la même liste de façon désordonnée :

essai / groupe	1	2	3	4
hiérarchie	73	106	112	112
désordre	21	39	53	70

Les expérimentateurs observent que ce rappel est très efficace puisque, dès le troisième essai, la totalité des 112 mots est rappelée ce qui, en soi, constitue une belle performance.

c) l'apprentissage par coeur

A la lumière de toutes ces informations, il semble intéressant d'aborder de façon concise la technique d'apprentissage dite « par coeur ». La notion d'organisation, décrite ci-dessus, y est en effet présente.

Des chercheurs ont montré que « l'apprentissage par coeur » diffère d'une simple répétition passive. En 1962, Tulving expérimente une liste de mots. Des étudiants consultent la liste puis rappellent les mots dont ils se souviennent. Il leur est ensuite loisible de reconsulter la liste. Cependant, à chaque consultation nouvelle, les mots se présentent dans un ordre différent. Tulving s'aperçoit que, malgré le changement d'ordre des mots, chaque sujet rappelle, d'essai en essai et de manière toujours plus stable, les mêmes séquences de mots.

Il en conclut que l'apprentissage a lieu par des groupes propres à chaque sujet mais variés selon une multitude de groupements. La répétition n'est donc pas le moteur de l'apprentissage, mais plutôt une fonction dont le but est de maintenir dans la mémoire à court terme les mots qui vont être reliés entre eux.

2) L'organisation verbale et l'organisation imagée

Le second mode d'organisation est l'organisation verbale. Pour Miller (Biblio 3), le langage est une organisation complexe de codes et offre évidemment de grandes possibilités d'organisation en mémoire. C'est pourquoi, comme nous l'avons déjà mentionné plus haut, Miller voit dans le langage une hiérarchie de niveaux de structures constituée de lettres, mots, phrases, scénarios, macrostructures. Nous allons, à présent, décrire le principe de l'organisation verbale en relatant une expérience à ce propos.

En 1974, Garten et Blick (Biblio 3) proposent une liste de couples de mots à mémoriser suivant trois conditions. Dans la première, les deux mots du couple, par exemple « microscope - bactérie », sont intégrés dans la phrase « le scientifique utilise le *microscope* pour étudier la *bactérie* ». Dans une deuxième

condition, souvent appelée médiation verbale, les deux mots sont reliés par un simple mot-clé : « laboratoire ». Enfin, la troisième condition est un groupe de contrôle où le couple est répété.

Le tableau ci-dessous décrit l'état de connaissance des couples en fonction de la condition et du délai :

Condition \ Délai de rappel	2 semaines	8 semaines
Phrase	75 %	55 %
Mot-clé	75 %	54 %
Répétition	55 %	38 %

Garten et Blick concluent que le rôle intégrateur de la phrase et du mot-clé est très efficace et persiste, même après plusieurs semaines. En revanche, l'absence de différence entre le rôle de la phrase et du mot-clé indique que la phrase a un rôle intégrateur uniquement parce qu'elle permet d'établir une relation en mémoire entre les deux mots à apprendre. Ceci semblerait justifier la pratique pédagogique qui consiste à souligner les mots-clés dans le texte ou à les rappeler en fin de chapitre.

L'organisation imagée est, quant à elle, très similaire. En effet, le principe consiste à se représenter l'intégration des deux mots d'un couple sous forme d'une image mentale, par exemple « un *billet* flottant sur une *rivière* » pour le couple « billet - rivière ». Ceci ressemble évidemment tout à fait à l'organisation verbale qui intègre les deux mots du couple dans une phrase.

3) Dans notre logiciel

a) groupements

Dans notre programme, le principe de la catégorisation se traduit par une notion de groupement. La possibilité est offerte à l'utilisateur de grouper plusieurs sous-listes sur base d'un thème commun (voir annexes, écran 10). Chaque groupement de sous-listes est identifié par un nom qui doit logiquement être caractéristique du thème propre au groupement.

Ce procédé de groupement implique, comme nous l'avons expliqué ci-dessus, une augmentation de la qualité et du rendement de l'apprentissage. En effet, en constituant lui-même les groupements, l'utilisateur donne une structuration à la liste. Il établit des relations entre les sous-listes. L'apprentissage en est facilité.

Lors de l'apprentissage proprement dit, deux alternatives sont laissées au choix de l'utilisateur. Tout d'abord, l'ordinateur est capable de tenir compte des groupements constitués dans la liste. Dans ce cas, le programme d'apprentissage traite le premier groupe de façon exclusive. Ensuite, dès que l'utilisateur a acquis une bonne connaissance de ce groupe, l'ordinateur passe à l'apprentissage du groupe suivant. Tous les groupes sont ainsi passés en revue, pour terminer par les sous-listes qui n'appartiendraient éventuellement à aucun groupement. Si, par contre, l'utilisateur ne souhaite pas tenir compte des groupements, les sous-listes sont étudiées dans l'ordre où elles ont été encodées.

La seconde alternative concerne la possibilité de réviser les groupements définis auparavant. Concrètement, l'ordinateur présente le nom du groupement; l'utilisateur doit restituer toutes les sous-listes qui appartiennent à ce groupement. Il est loisible à l'utilisateur de recommencer ce traitement tant qu'il commet des erreurs. Ce principe est valable pour tous les groupes. Ainsi, l'utilisateur vérifie la maîtrise de la structuration de la liste étudiée (voir annexes, écran 18).

Cette méthode, nous semble-t-il, est bénéfique pour l'utilisateur. En effet, les noms des groupements présentés jouent le rôle des indices catégoriels dont l'utilité a été décrite ci-dessus. De plus, si toutes les sous-listes de la liste étudiée appartiennent à des groupements, la structuration constituée par les noms de ces groupes peut être considérée comme un plan de récupération pour l'utilisateur (voir page 50).

Le principe de cette fonctionnalité de groupement est identique dans tous les algorithmes d'apprentissage que nous avons mis au point.

b) images mentales

Le mode d'organisation verbale ou imagée que nous avons explicité ci-dessus est également utilisé dans notre logiciel. Pour chacun des items d'une sous-liste, ainsi que pour la sous-liste elle-même, nous avons prévu la possibilité d'encoder ce que nous appelons une association mentale ou image mentale (voir annexes, écran 11). Concrètement, une image mentale est un mot ou une phrase décrivant la représentation que l'on se fait de la sous-liste ou de l'item de la sous-liste étudiée. L'image mentale d'une sous-liste est, par exemple, la représentation que l'on se fait de la relation existant entre tous les items de la sous-liste.

Lors de l'encodage ou lors de l'apprentissage, il est encore possible d'associer les images mentales aux sous-listes. Elles seront consultées pendant l'apprentissage, lorsque l'utilisateur souhaitera obtenir une indication sur l'item de la sous-liste étudiée. Le principe que nous venons de décrire est utilisé dans tous les algorithmes d'apprentissage que nous avons mis au point.

Par Philippe Vignaux

3.3.2. Description détaillée du didacticiel

Quels que soient la longueur et le type des sous-listes, les différents algorithmes que nous avons mis au point sont essentiellement basés sur le principe de l'apprentissage par répétition. Comme nous l'avons vu précédemment, la répétition vise à maintenir le plus longtemps possible, en mémoire à court terme, les items que l'on mémorise pour que des liens et des associations puissent se créer entre eux. Ainsi, les répétitions successives amènent les items en mémoire à long terme et augmentent les chances qu'ils ont d'y rester.

Dans une moindre mesure, ces algorithmes sont directement basés sur le principe de l'apprentissage par création d'associations entre les items à mémoriser. C'est le cas, entre autres, pour l'utilisation des images mentales et des groupements qui créent des associations entre plusieurs sous-listes ou entre les items d'une même sous-liste.

3.3.2.1. Les paramètres d'apprentissage

Comme nous l'avons vu précédemment, nous avons mis au point plusieurs types d'algorithmes en fonction des différentes catégories de listes que l'utilisateur est amené à assimiler. Cependant, dans tous les cas, l'utilisateur aura la possibilité, avant chaque apprentissage, de déterminer un certain nombre de paramètres destinés à guider le déroulement de cet apprentissage (voir annexes, écran 12). Nous allons à présent décrire ces paramètres.

1) Information sur l'apprentissage précédent

Cette information renseigne l'utilisateur sur le type du dernier apprentissage de la liste sélectionnée. Comme nous l'avons mentionné auparavant (voir p.40), le type d'apprentissage est passif (l'utilisateur sélectionne les solutions parmi une liste de propositions) ou actif (restitution des solutions par encodage). Si aucun apprentissage n'a été effectué, l'utilisateur en est évidemment averti.

2) Type et fréquence d'apprentissage

Ce procédé permet à l'utilisateur de préciser si l'apprentissage désiré est passif ou actif. Il permet ensuite de déterminer la fréquence d'apprentissage, c'est-à-dire le nombre de fois qu'il devra répéter correctement une sous-liste avant qu'elle ne soit considérée comme connue (voir annexes, écran 13). L'utilisateur a, enfin, la possibilité de réviser une liste étudiée précédemment.

3) Nombre de sous-listes par écran

Comme l'intitulé l'indique, ce paramètre détermine le nombre de sous-listes que l'ordinateur affiche simultanément à l'écran lors d'un apprentissage. Ce nombre dépend évidemment de la taille des sous-listes : on peut, par exemple, afficher sept sous-listes de deux items alors qu'on affiche seulement deux sous-listes de dix items. Ce nombre dépend également du résultat du test de l'empan que l'utilisateur a effectué auparavant. Notre programme offre à l'utilisateur la possibilité de faire varier ce paramètre dans une fourchette allant de une à sept sous-listes par écran.

4) Mot-maître

Dans notre logiciel, la notion de mot-maître désigne, lors de l'apprentissage, l'item de la sous-liste que l'utilisateur souhaite recevoir de l'ordinateur. Les autres items de la sous-liste doivent alors être restitués par l'utilisateur.

Le choix du mot-maître est fonction du format de la liste étudiée. Par exemple, si la liste est constituée de sous-listes de deux items, l'utilisateur choisit comme mot-maître, soit le premier item, soit le second, soit les deux. Dans ce dernier cas, l'ordinateur fournit alternativement le premier et le second item de la sous-liste si bien que celle-ci est assimilée dans les deux sens.

5) Traitement des groupes

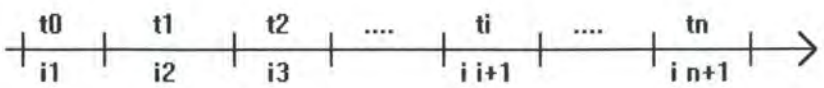
A l'intérieur d'une liste existent des groupements de plusieurs sous-listes. Grâce à ce paramètre, l'utilisateur détermine s'il tient compte de ces groupements lors de l'apprentissage. Il peut également, lors de l'apprentissage, déterminer s'il souhaite tester la connaissance qu'il a des sous-listes composant chacun des groupements (ce principe est décrit aux pages 53 et 63).

3.3.2.2. L'algorithme d'apprentissage des couples

Le principe de base de l'algorithme d'apprentissage des sous-listes de deux items est de montrer des couples à l'utilisateur puis de lui demander de les répéter correctement plusieurs fois, jusqu'à ce qu'ils soient entrés dans la mémoire à long terme.

Par conséquent, lors d'une séance d'apprentissage, un couple donné doit être répété plusieurs fois. Comme nous l'avons expliqué plus haut, l'utilisateur détermine la fréquence de répétition des couples de sa liste (appelée fréquence d'apprentissage ,voir page 56). Une de nos préoccupations fondamentales fut de répartir équitablement, lors du processus d'apprentissage, les différentes répétitions du couple. Par exemple, si l'utilisateur désire répéter chaque couple deux fois, le laps de temps séparant les deux apparitions de ce couple diffère du temps d'apparition avec une fréquence de cinq.

Nous avons ainsi divisé le processus d'apprentissage en une succession d'itérations. A chaque itération, l'ordinateur affecte un certain nombre de couples à traiter. La succession des itérations s'assimile donc à un parcours chronologique sur une ligne du temps :



t i : i ème intervalle de temps
i i : i ème itération

Le processus s'arrête lorsque tous les couples de la liste ont été étudiés et qu'ils ont été répétés correctement autant de fois que l'utilisateur l'a souhaité. Supposons, par exemple, qu'un couple apparaisse lors de l'itération numéro un. Si l'utilisateur se trompe lors de la restitution, le couple est à nouveau proposé lors de l'itération numéro deux. Par contre, si l'utilisateur ne s'est pas trompé, le couple n'est reproposé qu'à l'itération numéro quatre.

Dans la pratique, nous devons donc associer à chaque couple deux informations :

1) la fréquence de restitution correcte du couple lors du processus d'apprentissage. Nous ferons référence à cette information en terme de l'objet « affilée ».

2) le numéro de l'itération lors de laquelle le couple doit être proposé à l'utilisateur. Nous ferons référence à cette information en terme de l'objet « itération ».

Le terme « affilée » sous-entend, bien entendu, que les différentes répétitions d'un même couple par l'utilisateur ne sont pas consécutives. Ce terme désigne une totalisation des répétitions correctes du couple jusqu'à ce qu'elles soient interrompues par une répétition erronée. Ces répétitions se font donc lors d'itérations différentes.

A partir de ces deux informations, en tenant compte de la fréquence d'apprentissage, l'ordinateur gère et planifie les différentes apparitions de chaque couple en se référant à la matrice suivante dont nous illustrerons l'utilité à l'aide d'un exemple :

"affilée" / fréquence d'apprentissage	0	1	2	3	4	5	6
1	2	-1					
2	2	3	-1				
3	2	2	5	-1			
4	2	2	3	5	-1		
5	2	2	3	4	5	-1	
6	2	2	3	4	5	6	-1

Afin de parfaire la connaissance acquise, il se peut que l'utilisateur veuille répéter deux fois correctement chaque couple. Si ces deux répétitions sont couronnées de succès, l'utilisateur estime que les couples sont suffisamment connus pour qu'on les considère intégrés dans la mémoire à long terme. L'utilisateur choisira donc, dans ce cas-ci, une fréquence d'apprentissage de deux. Nous nous référerons ainsi exclusivement à la seconde ligne de la matrice.

Le couple en question est affiché à l'écran (en compagnie d'autres couples, nous y reviendrons par la suite) et disparaît ensuite. On montre alors à l'utilisateur l'un des items du couple et on lui demande de restituer l'item manquant. Deux situations sont évidemment possibles : soit la réponse est correcte, soit elle ne l'est pas.

Si la réponse est correcte, on augmente l'objet « affilée » d'une unité propre au couple en question. Cet objet indique que, à présent, le couple traité a été restitué correctement une seule fois. Le programme doit alors déterminer à quel moment le second et dernier rappel du couple aura lieu. Puisque le couple a bien été restitué, il serait logique d'attendre un peu plus longtemps avant de procéder à cette seconde répétition. Le couple réapparaîtra en fait trois itérations plus loin, c'est-à-dire lors de l'itération numéro quatre. Ce chiffre trois a été trouvé dans la matrice en prenant l'intersection de la seconde ligne (fréquence = 2) et de la colonne dont l'indice est déterminé par la valeur de l'objet « affilée », c'est-à-dire 1. Ainsi, entretemps, d'autres couples sont proposés à l'utilisateur; lors de l'itération numéro quatre, le couple réapparaîtra pour la seconde fois. S'il est à nouveau bien restitué, la valeur de l'objet « affilée » sera 2 et la valeur correspondante dans la matrice sera -1, pour indiquer que le couple sort du processus d'apprentissage et entre dans la mémoire à long terme. A ce moment, il aura été répété correctement deux fois, comme souhaité.

Si la réponse est incorrecte, la valeur de l'objet « affilée » est réinitialisée à zéro, ce qui témoigne bien du fait que, à ce moment, la série de réponses correctes est terminée. La valeur correspondant, dans la matrice, à la seconde ligne et à la colonne d'indice 0 est 2. Aussi, après avoir dévoilé la réponse correcte, le couple revient lors de l'itération numéro 1+2=3 et doit donc à nouveau être restitué d'affilée deux fois correctement.

Si l'utilisateur souhaite répéter cinq fois le couple lors du processus d'apprentissage, la ligne de la matrice à observer est la cinquième. Dans ce cas, il s'écoule respectivement deux, trois, quatre et cinq itérations entre chaque rappel du couple. Ce système permet ainsi à l'utilisateur d'attendre progressivement plus longtemps entre chaque rappel des couples étudiés.

Nous devons encore préciser que cette matrice, qui détermine le renvoi d'un couple d'une itération à une autre, a été constituée de façon arbitraire. Elle a ensuite été modifiée de nombreuses fois suite à l'utilisation du programme. Bien que sa constitution actuelle nous semble très efficace, rien n'empêcherait, à l'avenir, d'accorder à un utilisateur averti la possibilité de modifier cette matrice pour, éventuellement, allonger certains délais entre les rappels des couples. La durée d'une itération dépend évidemment du nombre de couples.

A présent, il est utile de donner quelques éclaircissements sur la façon dont l'algorithme parcourt la liste à étudier. Initialement, les dix premiers couples de la liste sont traités lors de la première itération. Ils sont proposés à l'utilisateur à raison de sept couples maximum par écran (voir page 56). Les couples appartenant à une itération donnée pourront donc occuper plusieurs écrans successifs.

Après avoir montré les couples d'un écran, l'ordinateur interroge l'utilisateur. Chaque couple est traité individuellement. Le programme ne montre plus que le mot-maître et demande à l'utilisateur d'encoder l'item manquant. Lors de l'encodage des solutions, les couples sont traités dans un ordre différent par rapport à l'ordre de présentation, afin d'empêcher l'utilisateur de retenir machinalement la succession des solutions.

Pour la correction des couples présentés, l'ordinateur détermine le numéro de l'itération à laquelle chacun des couples devrait réapparaître. Ce calcul se fait, comme nous l'avons décrit ci-dessus, en fonction de la matrice, de la fréquence d'apprentissage et, bien entendu, en fonction de la validité de la réponse. La correction terminée, l'ordinateur vérifie si les couples ont été répétés avec succès autant de fois que le stipule la fréquence d'apprentissage. Dans ce cas, ces couples sortent du processus d'apprentissage et, pour autant que la liste à étudier ne soit pas épuisée, sont remplacés par d'autres couples.

Par conséquent, lors de chaque nouvelle itération, deux catégories de couples sont traitées : d'une part, les couples qui ont déjà été étudiés précédemment et qui réapparaissent lors de cette itération; d'autre part, une quantité de nouveaux couples égale à la quantité de couples ayant quitté le processus d'apprentissage lors de l'itération précédente.

Les couples entrés dans la mémoire à long terme ne seront plus étudiés par la suite. Ainsi, si l'utilisateur interrompt l'apprentissage en cours de processus, les mots mémorisés ne sont plus chargés par l'algorithme lors d'un nouvel apprentissage. Ainsi, l'utilisateur répartit sur plusieurs jours l'apprentissage d'une liste assez longue. Remarquons encore que si toutes les sous-listes ont déjà été étudiées précédemment et que l'utilisateur choisit l'option d'apprentissage, l'ordinateur refuse d'exécuter ce processus et oblige l'utilisateur à réviser sa liste. De plus, certains items, soi-disant entrés dans la mémoire à long terme, sont oubliés avec le temps. C'est ce que détecte le processus de révision (voir page 71).

Les principes décrits ci-dessus s'appliquent à l'étude des couples avec mot-maître. L'apprentissage se faisait dans un seul sens (l'utilisateur restitue le mot qui n'est pas désigné comme maître). Il peut cependant arriver que l'utilisateur souhaite étudier les items des sous-listes dans les deux sens, auquel cas il désigne deux mots-maîtres : le premier et le second item du couple. Nous avons donc mis au point un processus permettant un tel apprentissage. Les deux items du couple sont ainsi présentés alternativement à l'utilisateur. Chacun des deux items est restitué autant de fois que la fréquence d'apprentissage l'exige, selon le processus décrit ci-dessus. La particularité de ce processus est que l'apprentissage a lieu simultanément dans les deux sens.

L'algorithme d'apprentissage des couples tient évidemment compte des groupements qui peuvent éventuellement exister au sein de la liste. Comme cela a été expliqué plus haut (voir page 48), chaque groupement est étudié séparément. Ensuite, l'utilisateur peut contrôler la connaissance acquise de la structuration de la liste en restituant, à partir du nom du groupement, les sous-listes appartenant à ce groupement.

En ce qui concerne l'utilisation des images mentales, bien que le principe ait également été décrit plus haut, il nous semble utile de décrire leur contrôle par l'algorithme. Les images mentales visent à donner à l'utilisateur l'occasion de créer des associations lui permettant de retrouver plus facilement l'item d'une sous-liste ou la sous-liste qu'il souhaite mémoriser. Lors du processus d'apprentissage, lorsque l'utilisateur ne trouve pas l'item manquant du couple, il peut faire appel à l'image mentale associée à l'item recherché, ce qui devrait l'aider à trouver la solution. Si l'utilisateur qui demande l'image mentale associée à un item du couple ne peut, malgré tout, fournir la réponse attendue, alors l'objet "affilée", contrôlant pour chaque couple le nombre de restitutions correctes, est remis à 0 et l'utilisateur doit recommencer la série de réponses correctes. Si, par contre, après un recours à une image mentale, l'utilisateur délivre une réponse correcte, l'objet "affilée" n'est ni incrémenté, ni décrémenté. En effet, le recours à l'image mentale, bien qu'il soit utile et avantageux, ne peut être considéré à lui seul comme un moyen de retenir un item. L'utilisateur est ainsi obligé, pour avancer dans la série de réponses correctes, de se souvenir de l'item. L'utilisation d'une image mentale ne l'avantage pas et ne le pénalise pas. Précisons que si le principe des images mentales n'existait pas, l'utilisateur devrait, à chaque erreur, recommencer la série de réponses correctes, exercice très pénalisant surtout dans le cas d'une grande fréquence d'apprentissage.

Cet exemple illustre notre propos : supposons qu'un étudiant soit amené à étudier la traduction française de mots anglais. Il devra savoir que le mot anglais « green » se traduit en français par le mot « vert ». Afin de se donner une aide supplémentaire, il associe au mot « vert » l'image mentale suivante : « couleur de l'espoir » parce que l'association « vert - couleur d'espoir » est établie de façon claire dans son esprit.

Lorsqu'on lui soumettra le mot « green », le sujet pourra faire appel à l'image mentale et, comme on peut aisément le concevoir, il se rappellera plus facilement la réponse correcte. Cependant, si nous n'y prenons garde, l'utilisateur peut faire systématiquement appel à l'image mentale, ce qui lui simplifie considérablement la tâche. L'image mentale aide l'utilisateur à trouver la solution. Cependant, l'idéal est de parvenir à une mémorisation sans recours à cet adjuvant. C'est pourquoi nous avons choisi, lorsqu'une réponse correcte est fournie après recours à l'image mentale, de ne pas totaliser cette réponse. En somme, si l'utilisateur choisit dans son processus d'apprentissage de répéter correctement trois fois chaque couple, il devra nécessairement fournir trois répétitions correctes sans avoir utilisé l'image mentale.

Lorsque le processus d'apprentissage est terminé, il est essentiel d'informer l'utilisateur de la qualité et de la quantité de son apprentissage. Apparaissent alors trois informations. La première information donne à l'utilisateur une indication sur le pourcentage de couples étudiés dans la liste. Rappelons que l'apprentissage d'une liste peut se faire en plusieurs séances. Le pourcentage couvre donc la totalité de ces séances. La seconde information indique le nombre de couples étudiés lors de l'apprentissage en cours. Enfin, la troisième information indique le nombre de couples qui doivent encore être étudiés dans la liste.

3.3.2.3. L'algorithme d'apprentissage des sous-listes de taille supérieure à 2

Cet algorithme a été développé afin de permettre l'apprentissage des sous-listes dont la taille est supérieure à deux items. Les sous-listes d'une liste étudiées avec cet algorithme doivent toutes être de la même taille. Cependant, l'algorithme est également utilisable pour l'étude des listes de taille variable, moyennant certaines restrictions que nous préciserons par la suite.

Nous allons dans un premier temps décrire de façon plus précise le fonctionnement de l'algorithme. Ensuite, nous justifierons la stratégie adoptée en expliquant notamment pourquoi nous ne souhaitons pas utiliser le principe de l'algorithme des couples dans ce cas.

Quelle que soit la taille des sous-listes, nous n'étudierons qu'une seule sous-liste à la fois. L'utilisateur devra donc obligatoirement être capable de restituer tous les items appartenant à la sous-liste traitée avant de passer à la suivante. La sous-liste étudiée est donc affichée à l'écran. Elle disparaît ensuite. Il est demandé à l'utilisateur de restituer les items dont il se souvient.

A ce stade, deux situations se présentent. D'un côté, l'utilisateur désigne un mot-maître lors de l'apprentissage de la liste. Dans ce cas, il s'agira indifféremment d'un des items de la sous-liste. D'un autre côté, l'utilisateur ne désigne aucun mot-maître. Si un mot-maître est désigné, on le montre à l'utilisateur et on attend que ce dernier restitue les items manquants. Si aucun mot- maître n'a été désigné, l'utilisateur doit restituer tous les items de la sous-liste.

Après la restitution, l'ordinateur procède à la correction des réponses de l'utilisateur. L'algorithme réaffiche alors à l'écran les items oubliés ou non restitués et interroge à nouveau l'utilisateur sur la sous-liste traitée, jusqu'à ce que l'utilisateur ait restitué correctement tous les items de cette sous-liste.

Si la sous-liste est de taille considérable (entre cinq et dix items), le nombre d'items à restituer devient malgré tout fastidieux, surtout dans la cas d'un apprentissage actif. Aussi avons-nous prévu qu'un item restitué deux fois correctement d'affilée soit proposé deux fois d'office à l'utilisateur qui continue, entretemps, à être interrogé sur les items non assimilés.

L'utilisateur consacre ainsi toute son attention aux items erronés, d'autant que l'expérience a montré une tendance à commettre les mêmes erreurs.

Lorsqu'une sous-liste est étudiée de cette façon, elle risque d'être oubliée assez vite si elle n'est pas répétée. Cet oubli est notamment suscité par les interférences créées par les items des sous-listes intervenant par la suite dans le processus d'apprentissage. Il faudra donc rappeler plusieurs fois chacune des sous-listes. Le nombre des rappels est déterminé par le paramètre que nous avons appelé fréquence d'apprentissage (voir page 56).

Dans la pratique, nous avons utilisé un certain nombre d'ensembles (autant d'ensembles que de rappels souhaités). Chaque ensemble est susceptible de contenir un nombre déterminé d'items de sous-listes. Si l'utilisateur souhaite répéter cinq fois chaque sous-liste, l'algorithme utilise cinq ensembles. Ces cinq ensembles sont capables de contenir respectivement 20, 40, 80, 100 et 120 items (ces nombres sont purement arbitraires).

Lorsque l'utilisateur a terminé l'apprentissage d'une sous-liste, les items sont rangés dans le plus petit des ensembles. Par exemple, si la taille des sous-listes est de cinq items, on pourra ranger quatre sous-listes dans cet ensemble. Lorsque le premier ensemble est rempli, on révise alors les sous-listes appartenant à cet ensemble. Pour procéder à cette révision, on affiche le mot-maître si l'utilisateur en a désigné un. Si ce n'est pas le cas, on montre un bref instant les items de la sous-liste à restituer. En effet, il s'agit de la seule façon d'indiquer à l'utilisateur quelle sous-liste il est censé rappeler. Pour remédier à cet inconvénient, nous conseillerons ainsi à l'utilisateur de désigner un mot-maître, procédure beaucoup plus pratique en cas de révision des sous-listes.

Les sous-listes correctement révisées entrent dans l'ensemble de capacité immédiatement supérieure, c'est-à-dire celui pouvant contenir quarante items. Par contre, les sous-listes erronées quittent l'ensemble dans lequel elles se trouvent et rejoignent le processus d'apprentissage : elles devront être réétudiées puisque l'utilisateur ne les connaît plus parfaitement.

Le fonctionnement que nous venons de décrire s'applique également aux autres ensembles. Chaque fois que l'un de ces ensembles est rempli, on révise les sous-listes et on fait passer les sous-listes correctement restituées dans l'ensemble de capacité immédiatement

supérieure. Les sous-listes mettront alors progressivement plus de temps avant d'être rappelées puisque les ensembles dans lesquels elles entrent deviennent de plus en plus grands et mettent donc plus de temps avant d'être remplis.

D'autre part, il ne faut pas confondre le principe de rappel des sous-listes étudiées et l'algorithme général de révision. Le principe des rappels fait partie du processus d'apprentissage des sous-listes de plusieurs items, tandis que l'algorithme général de révision, qui sera décrit plus loin, s'applique, lui, à tous les types de listes.

L'algorithme que nous venons de décrire tient évidemment compte des groupements et des images mentales qui existent au sein de la liste étudiée. Nous ne décrirons plus le principe de leur utilisation et de leur gestion puisqu'il a déjà été exposé dans l'algorithme consacré à l'étude des couples.

L'algorithme, présenté ci-dessus, pourra également être utilisé dans le cas de l'apprentissage d'une liste constituée de sous-listes de taille variable. Le déroulement des opérations est rigoureusement similaire à ce qui a été décrit ci-dessus, à une exception près. En effet, l'utilisateur désigne comme mot-maître le premier item de la sous-liste (voir page 45). D'autre part, les ensembles auxquels nous avons fait référence ci-dessus se verront incrémentés d'un nombre variable d'items chaque fois qu'une sous-liste entrera dans ces ensembles.

Lors du rappel des sous-listes de taille variable, l'utilisateur recevra, s'il le souhaite, une indication sur le nombre d'items à restituer, puisque ce nombre varie d'une sous-liste à l'autre. Pour autant que l'utilisateur souhaite l'obtenir, cette information l'aidera dans la tâche de restitution des items d'une sous-liste.

Si nous avions réparti l'étude d'une sous-liste sur plusieurs itérations, comme c'est notamment le cas dans l'algorithme d'étude des couples, nous pensons que l'apprentissage de cette liste aurait été beaucoup plus difficile à réaliser, en tout cas beaucoup plus lent. En effet, supposons qu'on présente à l'utilisateur une sous-liste de huit items et qu'il en restitue correctement quatre lors du premier essai; supposons que, comme c'est le cas dans l'algorithme des couples, on renvoie l'étude de cette sous-liste à une itération ultérieure; entretemps, d'autres sous-listes sont soumises à l'utilisateur. Lorsque la sous-liste en

question réapparaît, il y a de fortes chances pour que les quatre items mémorisés la première fois aient été oubliés en raison des interférences créées par l'apprentissage des items des autres sous-listes.

En utilisant ce système, nous pensons que l'apprentissage serait de moins bonne qualité et qu'il serait en tout cas ralenti puisque l'utilisateur devrait réétudier des items par ailleurs connus. Evidemment, dans la technique utilisée, l'utilisateur risque également d'oublier une partie des items des sous-listes mémorisées. Nous pensons cependant que cet oubli sera moindre et que l'utilisateur réétudiera plus facilement une sous-liste déjà connue complètement.

Dautre part, si nous avions utilisé un produit similaire à l'algorithme des itérations, il aurait été plus difficile de répartir chronologiquement les différentes apparitions d'une sous-liste lors des différentes itérations. Rappelons que dans l'algorithme des couples, la durée d'une itération est déterminée par le nombre de sous-listes devant passer au cours de cette itération. Dans le cas qui nous occupe, ce délai de rappel est plus difficile à calculer puisque les sous-listes peuvent avoir des tailles différentes (tailles variables). De plus, dans la technique que nous avons mis au point, le rappel des sous-listes intervient lorsqu'un ensemble est rempli. Or, la taille d'un ensemble n'est pas déterminée par le nombre des sous-listes que cet ensemble contient, mais bien par un nombre maximal d'items de sous-listes. Ce nombre étant fixe, le rappel aura donc lieu à intervalles réguliers.

3.3.2.4. L'algorithme d'apprentissage des sous-listes à un seul item

Comme nous l'avons dit précédemment, nous estimons que ce type de sous-liste est le moins fréquemment utilisé. De plus, lorsque de telles listes devront néanmoins être étudiées, leur taille sera réduite et ne dépassera jamais vingt à trente items. Il est en effet insensé d'étudier des listes dont la taille est démesurément grande. De telles listes ne sont d'ailleurs d'aucune utilité pour l'utilisateur.

Les listes appartenant à ce type seront ordonnées ou désordonnées. Nous allons donc, tout d'abord, décrire le fonctionnement de l'apprentissage dans le cas où l'ordre des items n'a pas d'importance.

La liste à étudier est découpée en parties regroupant chacune sept items. Ce nombre est en effet la quantité moyenne d'articles qu'un sujet peut emmagasiner en une fois dans la mémoire à court terme. Ce nombre varie toutefois en fonction du résultat du test de l'empan des mots effectué par l'utilisateur.

L'ordinateur affiche le premier groupe de sept items à l'écran. Le programme les fait ensuite disparaître. On demandera alors à l'utilisateur de restituer les mots dont il se souvient. Ce processus est réitéré jusqu'à ce que l'utilisateur se souvienne de tous les items du groupe. Ce principe est en fait le même que celui décrit à la page 64.

Lorsque le premier groupe est mémorisé, l'algorithme procède de façon analogue à l'étude des autres groupes de sept items. Rappelons qu'il n'y aura jamais plus de quatre ou cinq groupes dans une liste.

Lorsque tous les groupes sont connus, l'ordinateur demande à l'utilisateur de restituer un maximum d'items. Sur l'ensemble de la liste, il en aura oublié une certaine quantité. Ces items mal mémorisés sont alors de nouveau soumis à l'utilisateur, par groupe de sept, comme cela a été décrit ci-dessus. L'algorithme demande ensuite à l'utilisateur de restituer à nouveau l'ensemble des items de la sous-liste. Ce principe est reconduit jusqu'à ce que l'utilisateur maîtrise la totalité des items de la liste.

Cependant, la liste à étudier peut être ordonnée. Comme précédemment, on découpe la liste en groupe de sept items. L'ordinateur présente chacun de ces groupes à l'utilisateur et l'oblige à restituer les items de chaque groupe en respectant l'ordre d'apparition lors de la présentation. Tant que l'utilisateur oublie les items d'un groupe ou qu'il ne les restitue pas dans l'ordre adéquat, on ne passe pas à l'étude du groupe suivant.

Lorsque tous les groupes ont été traités de cette façon, l'algorithme affiche la liste à l'écran en cachant, à partir du début, un item sur trois. Il demande alors à l'utilisateur de restituer les items manquants. Cette opération est répétée autant de fois que des erreurs subsistent. Ensuite, le programme agit de la même façon, mais en cachant un item sur trois à partir du deuxième item et, enfin, un item sur trois à partir du troisième item. A ce stade-ci, tous les items de la liste auront été parcourus.

Dans un second temps, l'ordinateur recommence le même processus en cachant, cette fois, deux items consécutifs et en laissant visible les deux suivants. Par la suite, il augmente systématiquement la taille des trous et affiche un nombre décroissant d'items, jusqu'à ce que l'utilisateur soit finalement capable de restituer la totalité des items de la liste.

L'avantage de ce système est, qu'au cours de l'apprentissage, l'utilisateur est aidé par les associations de contiguïtés entre les items de la liste. Ensuite, à mesure que la connaissance de la liste augmente, on réduit le nombre d'items affichés à l'écran. Les associations d'items de plus en plus grandes que l'utilisateur retient à force de répétition peuvent ainsi être facilement replacées dans les trous correspondants.

Comme dans tous les algorithmes présentés jusqu'ici, l'apprentissage peut être actif ou passif. L'apprentissage passif est d'une grande utilité dans le cas où il serait plus important, pour l'utilisateur, de retenir l'ordre des items plutôt que les items eux-mêmes. En effet, les items sont présentés dans le désordre en bas de l'écran et l'utilisateur les place au bon endroit pour reconstituer la sous-liste en utilisant la souris et en s'évitant ainsi un travail d'encodage assez fastidieux.

Les images mentales sont, comme toujours, accessibles lors de l'apprentissage. Elles constituent un moyen de retenir la liste, dans la mesure où chacune des images mentales est une phrase évoquant l'item recherché. Ces phrases s'enchaîneront de façon

intuitive pour l'utilisateur et constitueront une histoire. Le restitution des phrases de cette histoire rappellera ainsi à l'utilisateur chacun des items de la sous-liste dans l'ordre voulu. Cette technique est d'ailleurs connue sous le nom de *méthode des loci* (cfr. Biblio 4).

Le principe du groupement de plusieurs sous-listes (ici on parlera de groupement d'items puisque chaque sous-liste est constituée d'un seul item) est également utilisable, pour autant que la liste étudiée ne soit pas ordonnée. Dans ce cas, les items de la liste ne pourront évidemment pas être déplacés et devront être étudiés de façon contiguë.

A la fin de l'apprentissage, l'utilisateur sera, comme toujours, informé du résultat de son apprentissage.

3.3.2.5. L'algorithme général de révision

Les algorithmes d'apprentissage que nous venons de décrire permettent à l'utilisateur d'acquérir la connaissance de la liste qu'il étudie. Si cette liste n'est pas régulièrement révisée, les sous-listes étudiées seront progressivement oubliées et l'effet de l'apprentissage sera anéanti. Selon les chercheurs, l'importance de cet oubli est considérable puisqu'il est non seulement considéré comme la dégradation des souvenirs mais encore comme le résultat de processus dynamiques que sont les interférences. Celles-ci seront provoquées par les apprentissages ultérieurs de l'utilisateur.

C'est pourquoi nous avons mis au point un algorithme dont le but est de réviser toutes les sous-listes d'une liste étudiée afin de détecter celles qui ont été oubliées. Celles-ci seront ensuite réétudiées par l'utilisateur via l'algorithme d'apprentissage approprié au type de la liste. Ce nouvel apprentissage est plus rapide que l'apprentissage initial. De fait, il est plus facile de mémoriser quelque chose qui était déjà connu auparavant. Nous précisons que l'algorithme de révision mis au point s'applique à tous les types de listes décrits et que la révision d'une liste peut se faire selon le mode d'apprentissage actif ou passif (quel que soit le mode d'apprentissage de cette même liste).

Le procédé de révision est le suivant : l'algorithme affiche à l'écran un item de la sous-liste à réviser. Cet item sera celui qui avait été désigné comme mot-maître lors de l'apprentissage. L'utilisateur encode alors les items manquants de la sous-liste. Par exemple, dans le cas de l'étude de couples d'items, si l'utilisateur a désigné deux mots-maîtres (il étudie les couples d'items dans les deux sens), il pourra logiquement désigner comme mot-maître, lors de la révision, soit le premier item, soit le second, soit les deux, auquel cas chaque couple sera révisé deux fois : une première fois avec le premier item comme mot-maître et une seconde fois avec le second item comme mot-maître. Un couple est considéré comme connu lorsqu'il a été correctement révisé dans les deux sens.

Lors de la révision, nous avons décidé que l'utilisateur n'aurait plus accès aux images mentales ainsi qu'aux groupements portant sur certaines sous-listes. Ces informations avaient toute leur raison d'être lors de l'apprentissage puisqu'elles étaient destinées à aider l'utilisateur dans l'acquisition d'une bonne connaissance de la liste étudiée.

Lors de la révision, l'utilisateur est censé connaître parfaitement la liste. Il n'y a donc plus de raison pour qu'il dispose de ces informations. Afin de briser les liens de dépendance qui auraient pu se créer entre certaines sous-listes (l'utilisateur se souvient d'une sous-liste uniquement s'il a pu restituer la précédente), l'algorithme de révision ne parcourt pas la liste séquentiellement. Au contraire, il cherche chacune des sous-listes de façon aléatoire.

Lorsque la séance de révision est terminée, l'utilisateur reçoit les détails des résultats de cette révision. Ces informations portent sur le nombre de sous-listes à retenir dans la liste ainsi que le nombre de sous-listes correctement révisées.

4

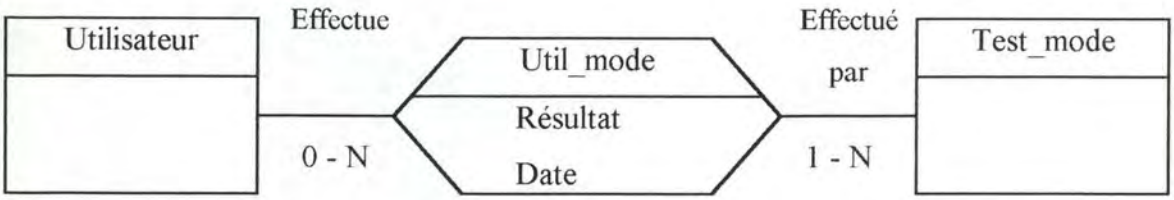
ANALYSE DU DIDACTICIEL

Ce chapitre contient l'analyse des différentes parties du didacticiel. Nous commencerons par l'analyse du test sur les modes de mémorisation (qui n'a pas été implémenté). Nous poursuivrons par l'analyse du test de l'empan de mémoire. Cette seconde partie a été entièrement implémentée. Nous terminerons par l'analyse de l'encodage des listes et de l'apprentissage des listes. Cette dernière partie a été partiellement implémentée.

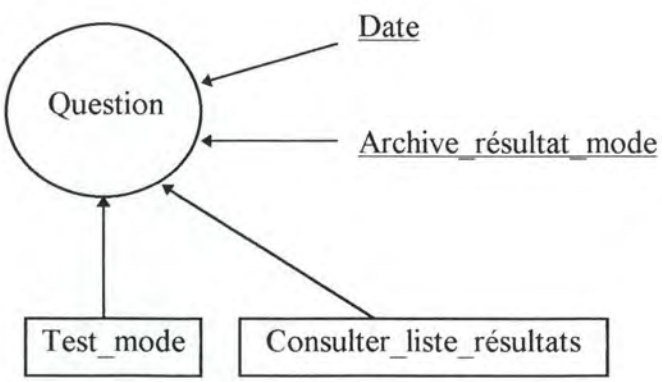
4.1. Test sur les modes de mémorisation

Par Valérie Walthéry

4.1.1. Schéma Entité - Association



4.1.2. Représentation des objets conceptuels



Légende :

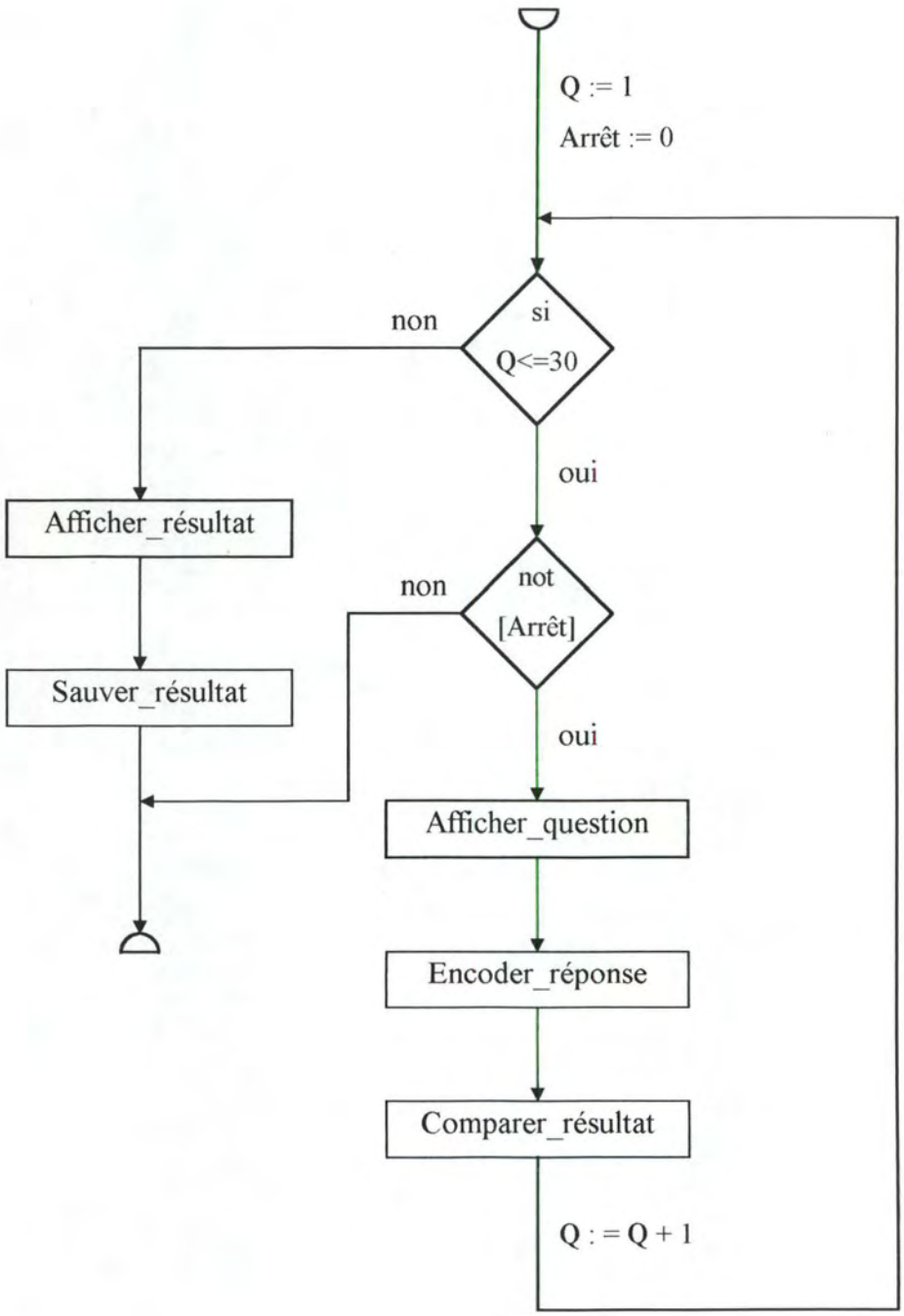
◯ : objet conceptuel

▭ : traitement

——— : donnée

4.1.3. Spécifications

4.1.3.1. Algorithme général « Test_mode »



Précondition : /

Postcondition : l'algorithme se termine lorsque l'utilisateur appuie sur le bouton « arrêt » ou lorsqu'il a répondu à toutes les questions.

4.1.3.2. Spécification de « Afficher_question »

Précondition : 1 <= Q <= 30

Postcondition : l'ordinateur affiche la question n° Q, se trouvant dans le fichier « Questions », à l'écran.

4.1.3.3. Spécification de « Encoder_réponse »

Précondition : /

Postcondition : l'utilisateur a répondu à la question.

4.1.3.4. Spécification de « Comparer_résultat »

Précondition : l'utilisateur doit avoir répondu à la question Q.

Postcondition : le résultat de l'utilisateur est comparé avec le tableau-type des résultats (se trouvant dans le fichier « tableau_type »). La variable qui représente la colonne contenant le résultat de l'utilisateur est incrémentée de 1. Par exemple, si l'utilisateur choisit la réponse « a » pour la question 1, la variable qui représente la colonne VII est incrémentée de 1 (le tableau-type se trouve à la page 30). Les résultats des questions, c'est-à-dire la valeur des variables représentant les sept colonnes du tableau-type, sont enregistrés dans le fichier « Résultat_mode ».

4.1.3.5. Spécification de « Afficher_résultat »

Précondition : l'utilisateur a répondu à toutes les questions.

Postcondition : les résultats du test sont affichés à l'écran avec un commentaire les concernant.

4.1.3.6. Spécification de « Sauver_résultat »

Précondition : le test doit avoir été entièrement réalisé.

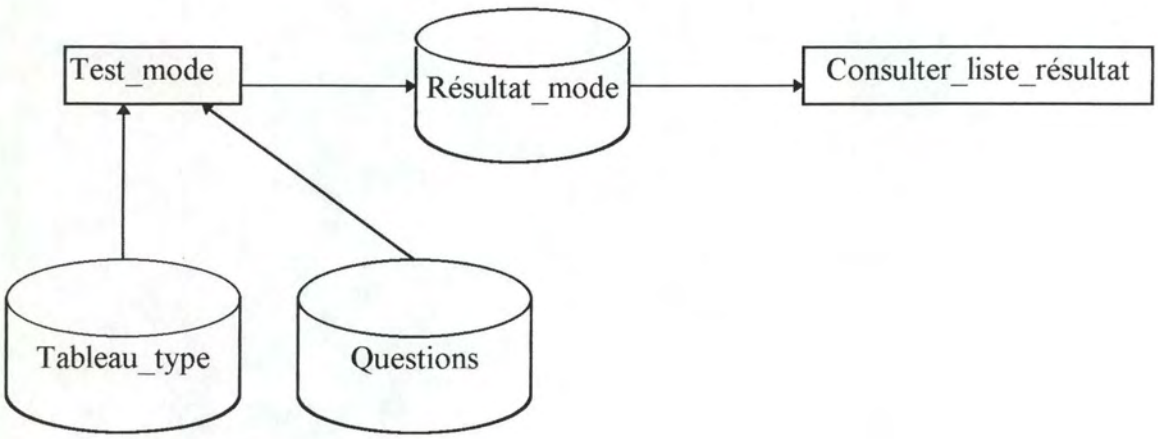
Postcondition : le résultat du test effectué et la date à laquelle il a été exécuté est enregistré à la suite des tests effectués précédemment.

4.1.3.6. Spécification de « Consulter_liste_résultats »

Précondition : /

Postcondition : affiche la liste des résultats et dates de tous les tests effectués par l'utilisateur.

4.1.4. Objets utilisés dans le logiciel



Légende :

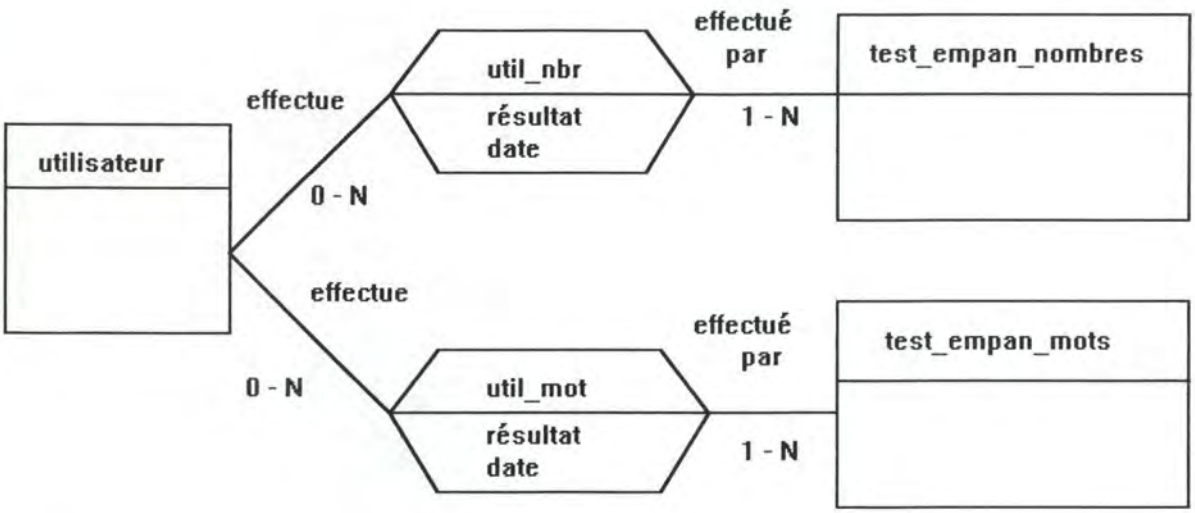
: traitement

: fichier

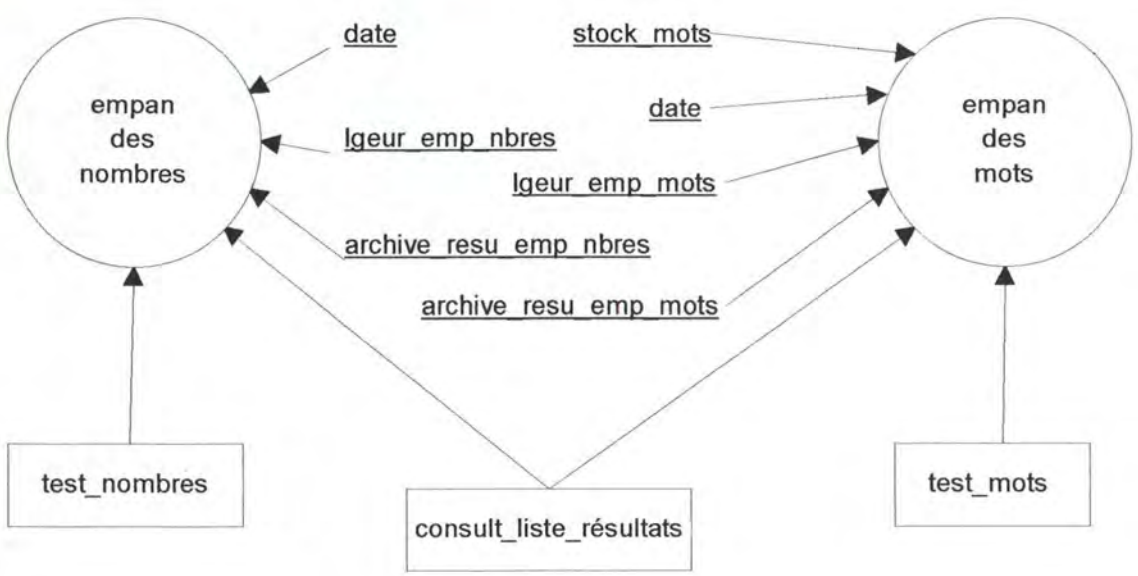
4.2. Exercices sur l'empan de mémoire

Par Philippe Vignaux

4.2.1. Schéma Entité - Association

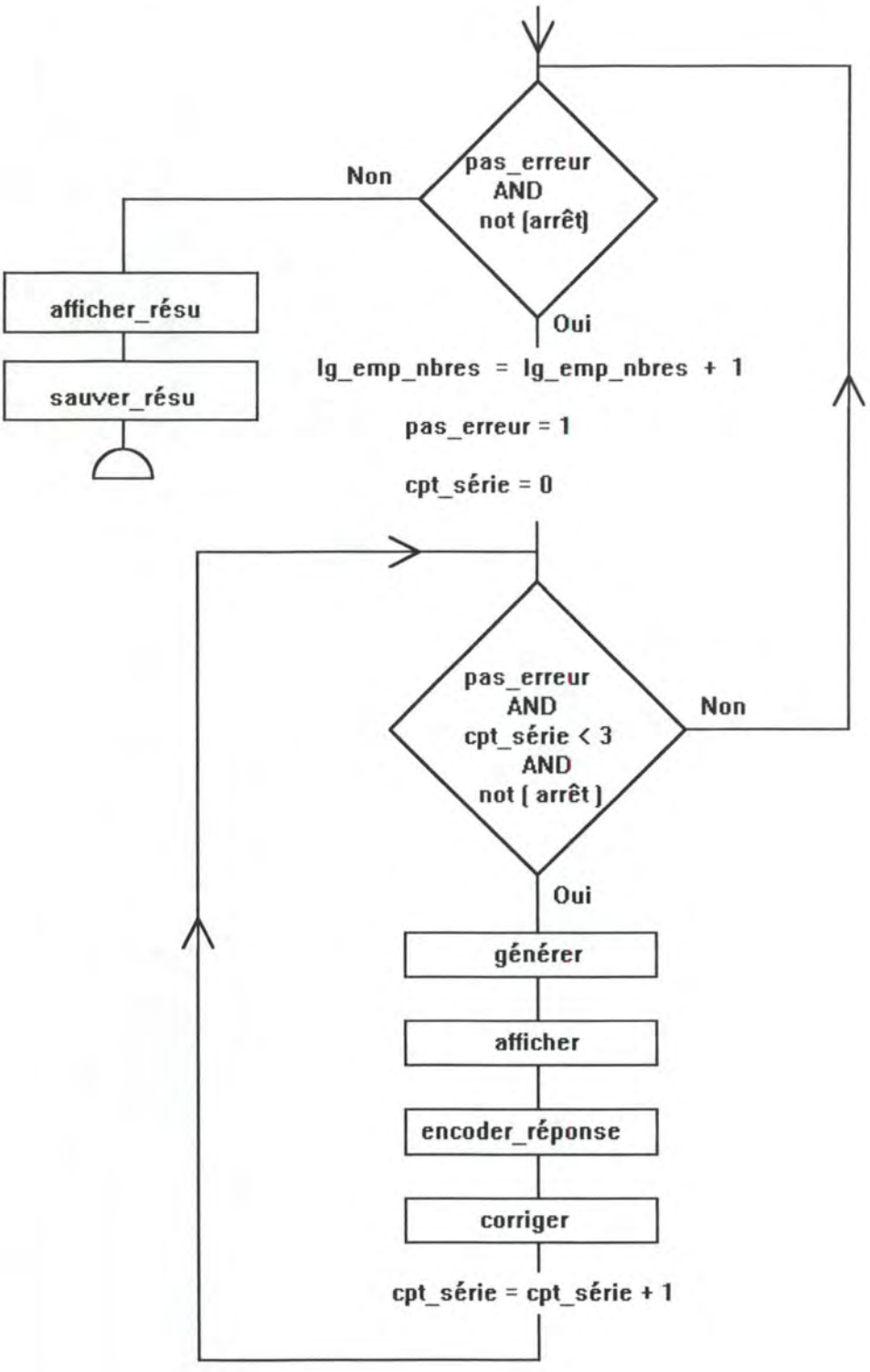


4.2.2. Représentation des objets conceptuels



4.2.3. Spécifications de « Test nombres »

4.2.3.1. Algorithme général "test_nombres"



Précondition : pas_erreur = 1
lg_emp_nbres = 3
arrêt = 0

Postcondition : l'algorithme se termine lorsque l'utilisateur appuie sur le bouton "arrêt" (arrêt = 1) ou lorsqu'il commet une erreur dans la séquence de chiffres qu'il encode.
lg_emp_nbres contient la taille des séquences de la dernière série de trois séquences de chiffres correctement encodée.

4.2.3.2. Spécifications de "générer"

Précondition : x = lg_emp_nbres

Postcondition : l'ordinateur génère x chiffres entre 0 et 9
la succession de ces chiffres constitue un nombre
solution = le nombre généré

4.2.3.3. Spécifications de "afficher"

Précondition : x = lg_emp_nbres

Postcondition : l'ordinateur affiche séparément chacun des x chiffres générés à raison de un chiffre par seconde.

4.2.3.4. Spécifications de "encoder_réponse"

Postcondition : l'utilisateur encode le nombre généré
rep = la réponse de l'utilisateur

4.2.3.5. Spécifications de "corriger"

Précondition : solution = le nombre généré par l'ordinateur
rep = la réponse de l'utilisateur

Postcondition : **si** rep = solution **alors** pas_erreur = 1
sinon pas_erreur = 0

4.2.3.6. Spécifications de "afficher_résu"

Précondition : lg_emp_nbres contient la taille des séquences de la dernière série de trois séquences de chiffres correctement encodée.

Postcondition : afficher lg_emp_nbres
afficher un commentaire sur le résultat de l'utilisateur

4.2.3.7. Spécifications de "sauver_résu"

Précondition : l'ensemble dans lequel on archive les résultats est constitué de n éléments (n >= 0).
chacun des éléments est le produit cartésien des informations
[date du test X résultat du test].

Postcondition : la date du test et le résultat du test effectué constituent le (n+1) ième élément de l'ensemble.

4.2.3.8. Spécifications de "consult_liste_résultats"

N.B. • Si l'utilisateur a effectué le test de l'empan des nombres, le traitement consulte les archives du test de l'empan des nombres (archive_résu_emp_nbres).

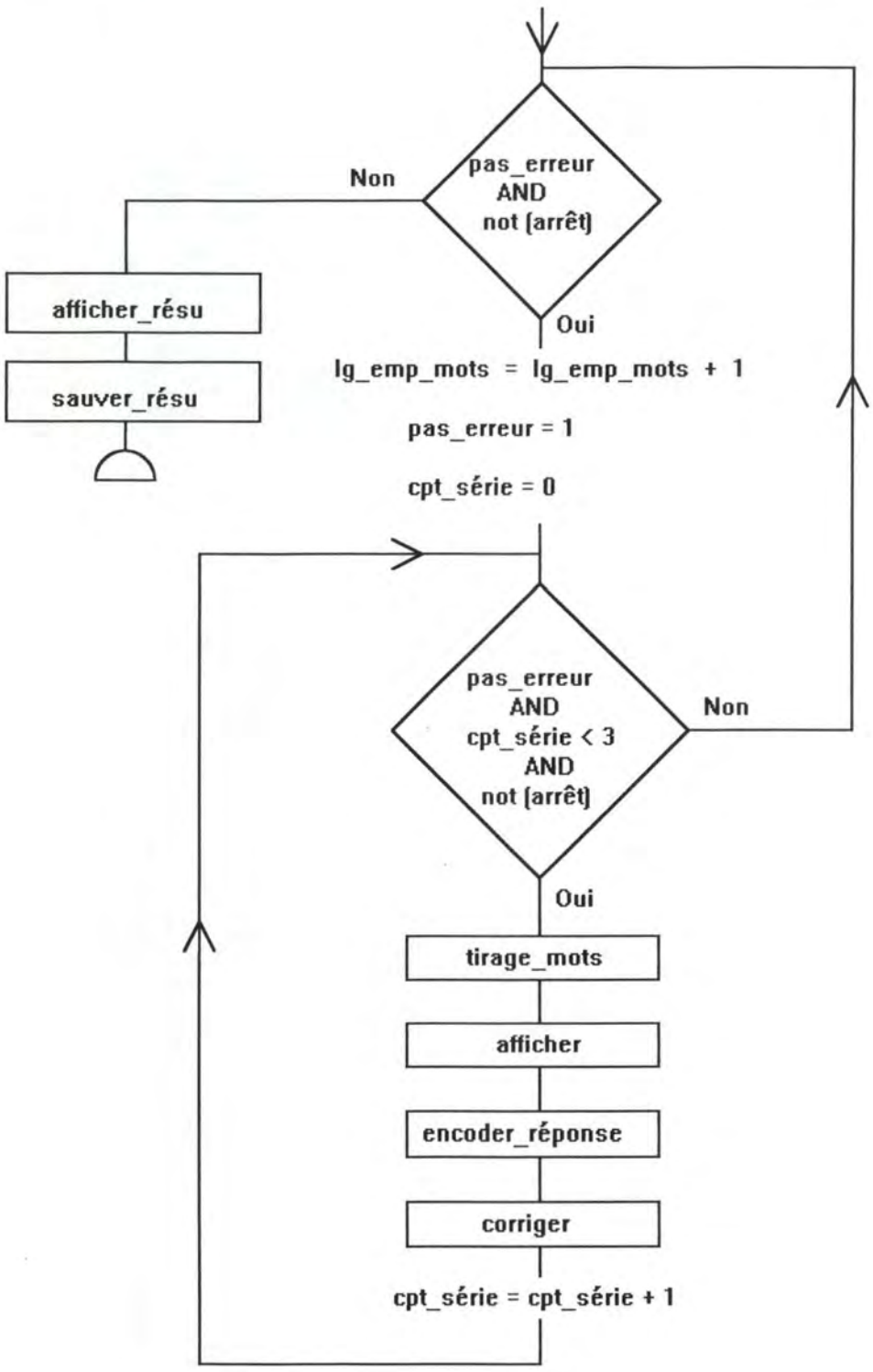
• Si l'utilisateur a effectué le test de l'empan des mots, le traitement consulte les archives du test de l'empan des mots (archive_résu_emp_mots).

Précondition : l'ensemble consulté est constitué de n éléments (n >= 0)

Postcondition : pour tout i : 0 <= i <= n , afficher (date du test **i)**
(résultat du test)**i**

4.2.4. Spécifications de « Test mots »

4.2.4.1. Algorithme général « Test_mots »



Précondition : pas_erreur = 1
lg_emp_mots = 3
arrêt = 0

Postcondition : l'algorithme se termine lorsque l'utilisateur appuie sur le bouton "arrêt" (arrêt = 1) ou lorsqu'il commet une erreur dans la séquence de mots qu'il encode.
lg_emp_mots contient la taille des séquences de la dernière série de trois séquences de mots correctement encodée.

4.2.4.2. Spécifications de "tirage_mots"

Précondition : x = lg_emp_mots
stock_mots est un ensemble constitué de n mots de langue française (n > 0)

Postcondition : l'ordinateur génère x chiffres entre 0 et n
pour chaque chiffre généré, on sélectionne le mot dont la position dans l'ensemble correspond à ce chiffre

4.2.4.3. Spécifications de "afficher"

Précondition : x = lg_emp_mots

Postcondition : l'ordinateur affiche séparément chacun des x mots sélectionnés à raison de un mot toutes les deux secondes

4.2.4.4. Spécifications de "encoder_réponse"

Postcondition : l'utilisateur encode la séquence de mots apparus à l'écran

4.2.4.5. Spécifications de "corriger"

Précondition : solution = la séquence de mots sélectionnée par l'ordinateur
rep = la séquence de mots proposée par l'utilisateur

Postcondition : **si** rep = solution **alors** pas_erreur = 1
sinon pas_erreur = 0

4.2.4.6. Spécifications de "afficher_résu"

Précondition : lg_emp_mots contient la taille des séquences de la dernière série de trois séquences de chiffres correctement encodée.

Postcondition : afficher lg_emp_mots
afficher un commentaire sur le résultat de l'utilisateur

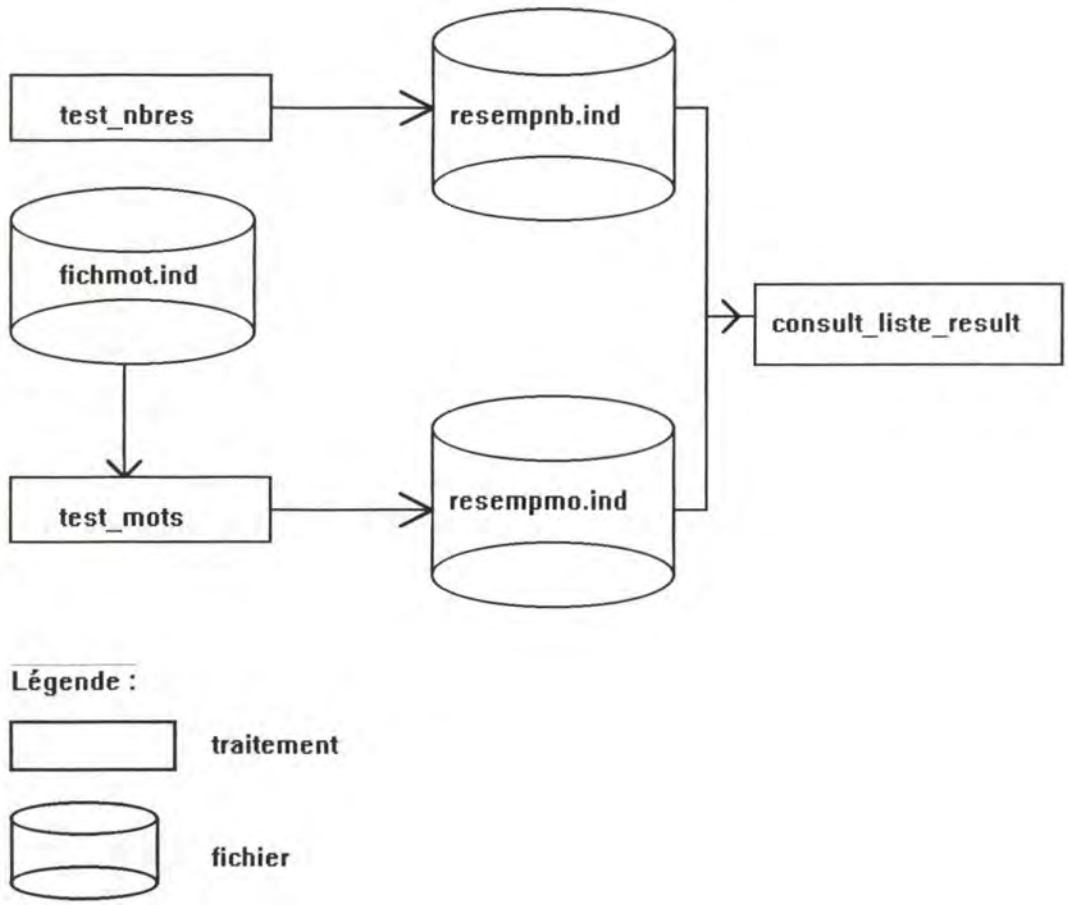
4.2.4.7. Spécifications de "sauver_résu"

Précondition : l'ensemble dans lequel on archive les résultats est constitué de n éléments (n >= 0).
chacun des éléments est le produit cartésien des informations
[date du test X résultat du test].

Postcondition : la date du test et le résultat du test effectué constituent le (n+1) ième élément de l'ensemble.

4.2.5. Objets utilisés dans le logiciel

4.2.5.1. Interaction entre traitements et ressources



4.2.5.2. Classes utilisées dans notre logiciel.

1) Classe "Tempnbre"

Cette classe contient les variables propres à l'exercice de l'empan des nombres. Nous y trouvons ainsi, par exemple, une variable faisant état de la longueur de l'empan des nombres ("lg_empan"), une variable contenant la solution correcte ("solution"), une variable contenant la réponse de l'utilisateur ("rep").

La classe contient également des fonctions accédant à ces variables : une fonction d'exécution du test proprement dit ("test_nbres") ainsi qu'une fonction destinée à sauver le résultat du test dans le fichier approprié ("sauver_result").

2) Classe "TD_Affemp"

Cette classe a pour but d'afficher séparément dans une boîte de dialogue chacun des chiffres du nombre généré. Elle est munie d'un chronomètre ("horloge") afin d'afficher un chiffre par seconde.

3) Classe "TD_Recemp"

Cette classe génère une boîte de dialogue afin de recevoir la réponse de l'utilisateur ("rep").

4) Classe "Tempmot"

Cette classe contient les variables propres à l'exercice de l'empan des mots. Nous y trouvons, par exemple, une variable faisant état de la longueur de l'empan des mots ("lg_empan"), un tableau contenant la séquence correcte de mots générés par l'ordinateur ("tab_emp_mot[20]"), un tableau contenant la séquence de mots proposées par l'utilisateur ("tab_rep_mot[20]").

La classe contient également des fonctions accédant à ces variables : une fonction d'exécution du test proprement dit ("test_nbres") ainsi qu'une fonction destinée à sauver le résultat du test dans le fichier approprié ("sauver_result").

5) Classe "TD_Affmot"

Cette classe a pour but d'afficher séparément dans une boîte de dialogue chacun des chiffres du nombre généré. Elle est munie d'un chronomètre ("horloge") afin d'afficher un mot par tranche de deux secondes.

6) Classe "TD_Recmot"

Cette classe génère une boîte de dialogue afin de recevoir la réponse de l'utilisateur ("rep").

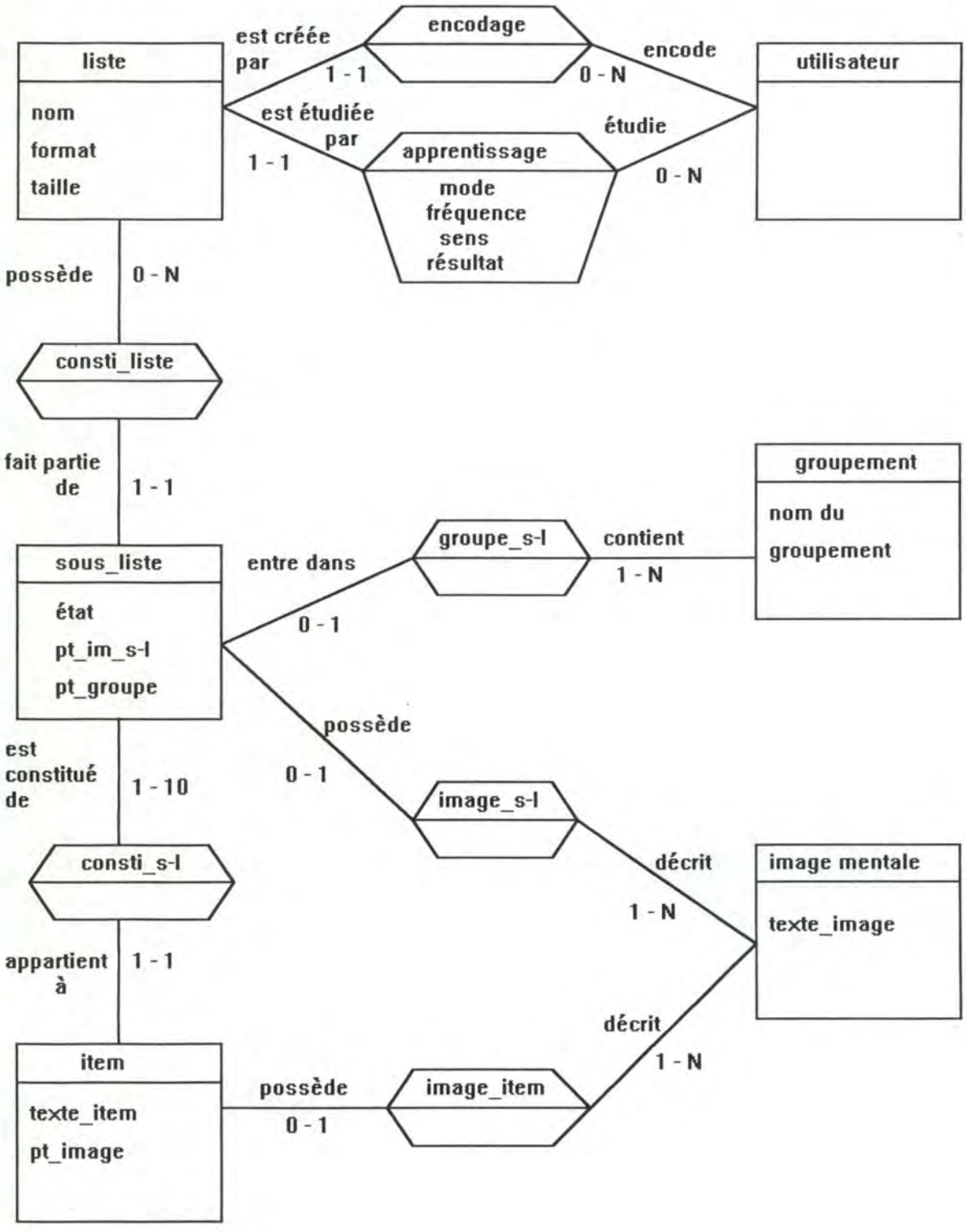
7) Classe "TD_Recmot"

Cette classe a pour effet de générer une boîte de dialogue visant à afficher les résultats archivés du test de l'empan des nombres ou du test de l'empan des mots (respectivement à partir des fichiers "resempnb.ind" et "resempnb.ind"). Les variables de cette classe sont essentiellement des variables de manipulations des fichiers (ex : "ptr", pointeur de fichier).

4.3. Encodage et apprentissage des listes

Par Philippe Vignaux

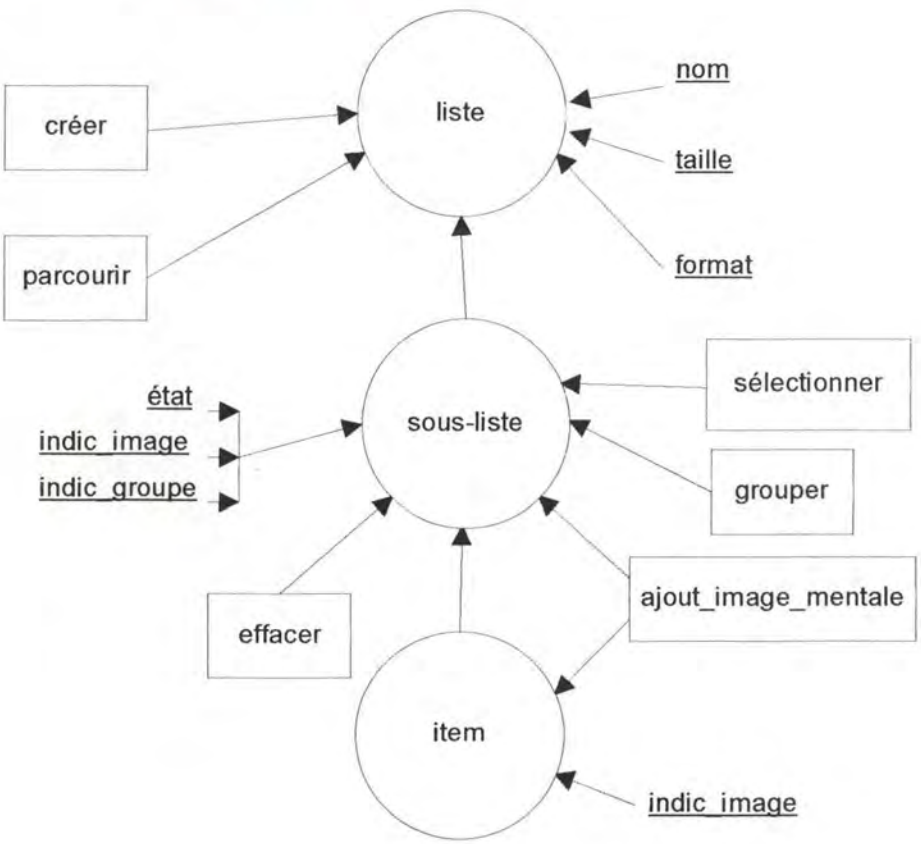
4.3.1. Schéma Entité - Association



4.3.2. Encodage des listes

Nous allons, dans un premier temps, décrire les fonctionnalités liées à l'encodage des listes. Par la suite nous décrirons celles relatives à leur apprentissage.

4.3.2.1. Représentation des objets conceptuels

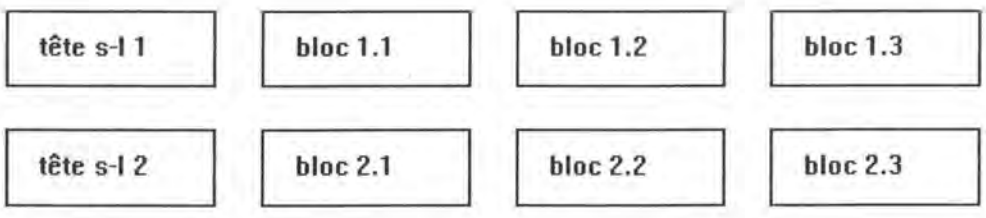


Nous allons exposer, ci-dessous, la manière dont les données sont structurées. Mais, auparavant, il nous faut signaler que :

- une liste est identifiée par son nom,
- le format d'une liste est soit fixe, soit variable,
- si une liste est de format fixe, les sous-listes devront obligatoirement avoir toutes le même nombre d'items, ce nombre étant déterminé par la taille choisie par l'utilisateur lors de la création de la liste,

- si une liste est de format variable, l'utilisateur détermine le nombre maximum d'items pouvant constituer chacune des sous-listes; ce nombre est la taille choisie par l'utilisateur lors de la création de la liste.

Chaque sous-liste sera constituée d'un bloc de tête suivi d'autant d'autres blocs qu'il y a d'items dans la sous-liste. Pour des sous-listes constituées de trois items, nous aurions la situation suivante :



...

1) Description des informations contenues dans un bloc de tête

Un bloc de tête contient trois informations :

état de la sous-liste	indicateur de présence d'image mentale	indicateur de groupement

a) état de la sous-liste : donne des informations sur l'état de la sous-liste. Il peut par exemple témoigner du fait que la sous-liste a été effacée. Dans ce cas, la sous-liste n'est plus prise en compte lors d'une édition à l'écran ou lors d'un apprentissage. Il peut aussi stipuler qu'une sous-liste doit encore être étudiée ou, au contraire, qu'elle a déjà été mémorisée avec succès, c'est-à-dire qu'elle est dans la mémoire à long terme de l'utilisateur et qu'elle ne doit plus être prise en compte lors d'un apprentissage, mais seulement lors d'une révision.

b) indicateur de présence d'image mentale : cet indicateur stipule qu'une image mentale a été associée à l'entièreté de la sous-liste.

c) indicateur de groupement : cet indicateur signale l'appartenance de la sous-liste à un groupement.

2) Description des informations contenues dans un bloc d'item

Un bloc de ce type contient deux informations :

item	indicateur de présence d'image mentale

a) item : cette information est l'un des items faisant partie de la sous-liste.

b) indicateur de présence d'image mentale : cet indicateur stipule qu'une image mentale à été associée à cet item de la sous-liste.

4.3.2.2. Description du fonctionnement

Toutes les sous-listes constituant une liste sont rangées dans un seul fichier. Toutes les images mentales qui font référence aux items ou aux sous-listes d'une liste sont également rangées dans un fichier propre aux images de cette liste. Il en va de même en ce qui concerne les noms des différents groupements existant au sein d'une liste.

Dans la pratique, il sera donc nécessaire de disposer d'un fichier de base (intitulé "nomliste.ind") reprenant, pour chacune des listes, son nom, le nom du fichier où sont rangées les sous-listes la constituant, le nom du fichier où sont rangées ses images mentales, le nom du fichier reprenant les intitulés des groupes existant au sein de la liste traitée et finalement, sa taille et son format.

Exemple illustrant le contenu du fichier « nomliste.ind » :

"nomliste.ind"

vocabulaire anglais	"fich1.ind"	"im1.ind"	"gp1.ind"	2	fixe
géographie	"fich2.ind"	"im2.ind"	"gp2.ind"	3	fixe

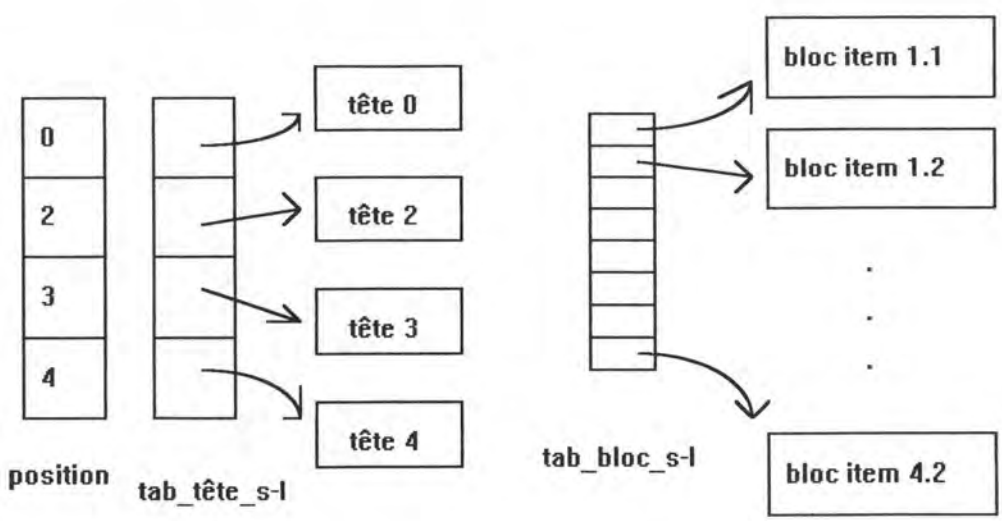
A partir de la représentation schématique d'une liste sous forme de fichier, nous allons illustrer son chargement en mémoire ainsi que les interactions entre cette liste et ses fichiers annexes (groupes, images). La liste utilisée dans l'exemple est celle décrite dans le fichier de base ci-dessus.

Représentation de "fich1.ind" sous forme de fichier :

tête			bloc d'item		bloc d'item	
état	point. gp.	pt. im. s-l.	item	pt. im. it.	item	pt. im. it.
0	1	-1	red	-1	rouge	-1
-1	-1	0	peace	-1	paix	-1
2	1	-1	green	-1	vert	1
0	2	-1	dog	-1	chien	-1
0	2	-1	mouse	2	souris	-1

. . .

Chargement de "fich1.ind" en mémoire :



On constate que, par rapport à la représentation de la liste dans le fichier, les sous-listes dont l'indicateur d'état est à '-1' ne sont pas chargées en mémoire.

Les valeurs potentielles de l'indicateur d'état sont :

'-1' : la sous-liste est effacée;
' 0' : la sous-liste doit encore être mémorisée;
' 2' : la sous-liste a été mémorisée avec succès.

Le tableau dénommé "tab_tête_s-l" stipule pour chacune des sous-listes chargées en mémoire la position occupée dans le fichier de rangement. La position qu'une sous-liste occupe en mémoire n'est pas nécessairement identique à celle qu'elle occupe dans le fichier puisque les sous-listes effacées ne sont pas chargées. Le tableau dénommé "tab_bloc_s-l" précise, quant à lui, où sont rangés tous les items de la liste.

Cette méthode d'organisation permet un mode de consultation uniforme des sous-listes, quelle que soit leur taille. Par exemple, pour accéder aux informations de la seconde sous-liste, il suffit de consulter l'élément du tableau "tab_tête_s-l" d'indice 1. Pour accéder aux items de la sous-liste, il suffit de multiplier cet indice par la taille de la sous-liste: 1*2=2. Dans le tableau "tab_bloc_s-l", les items de la seconde sous-liste sont donc rangés à partir de l'indice deux.

Lors de toute opération visant à modifier une sous_liste (rectification orthographique, entrée dans le long terme), l'indicateur d'état est positionné à une valeur supérieure à deux. Lorsque l'utilisateur quitte la liste, seules les sous-listes dont l'indicateur d'état est supérieur à deux sont réécrites dans le fichier à la position adéquate (tableau "position"). Les pertes de temps sont évitées car les sous-listes non modifiées ne sont pas sauvées.

Si l'on observe la figure représentant le fichier, on constate qu'une sous-liste, la seconde, ne fait partie d'aucun groupement. L'indicateur de présence de groupement est en effet positionné à la valeur '-1'. Pour d'autres sous-listes, cet indicateur est positionné tantôt à la valeur un, tantôt à la valeur deux. Les sous-listes s'intègrent donc respectivement dans un premier et dans un second groupement. Les appellations de ces groupements sont rangés dans un fichier dont le nom nous est fourni par le fichier de base (ce nom est "gp1.ind").

"gp1.ind"

couleurs
animaux

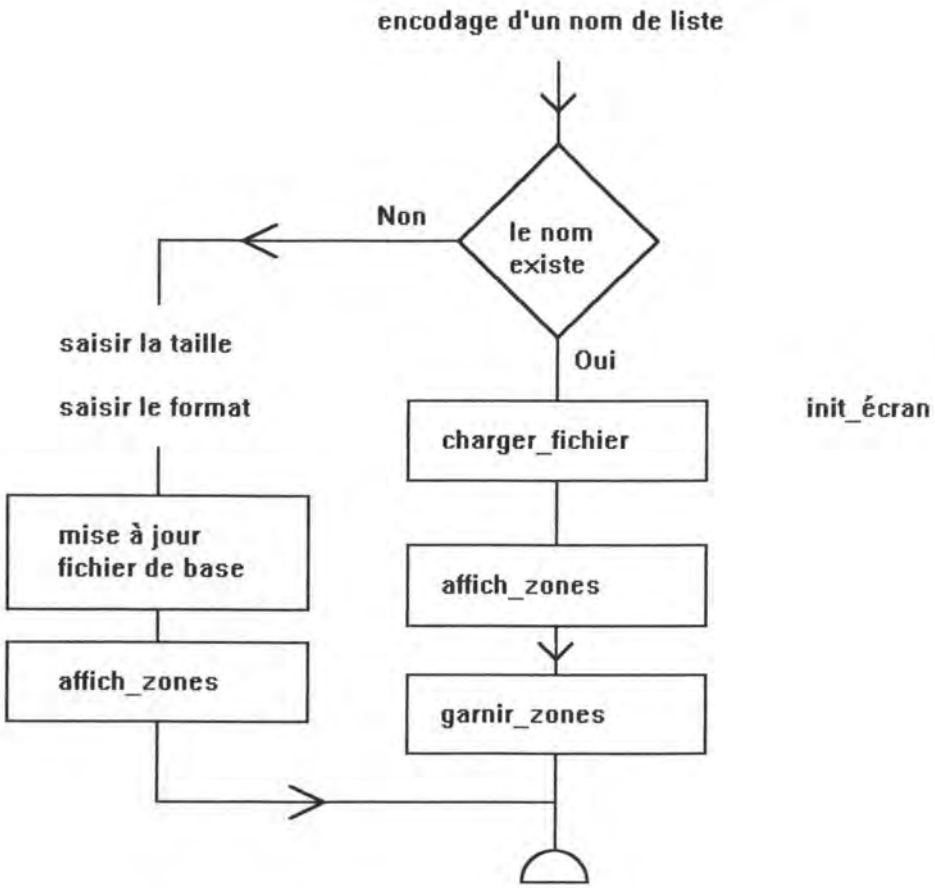
Pour accéder au nom d'un groupe de ce fichier, il suffit de lire l'enregistrement dont la position est donnée par la valeur de l'indicateur de présence de groupements. Par exemple, les sous-listes "dog-chien" et "mouse-souris" ont toutes deux un indicateur de présence de groupement ayant la valeur deux. Le second enregistrement du fichier des noms de groupes nous donne le titre "animaux".

Lorsque l'utilisateur souhaite grouper une sous-liste, les noms des groupements existant déjà lui sont présentés. Il peut alors sélectionner un de ces noms ou en encoder un nouveau. S'il souhaite grouper une sous-liste appartenant déjà à un groupement, le programme signale l'invalidité de cette opération. Selon notre hypothèse, une sous-liste ne s'intègre que dans un groupement.

Le principe de rangement des images mentales est identique à celui des groupes. Le fichier contenant les associations mentales est donné par le fichier de base (ici, "im1.ind"). La position du texte de l'image dans ce fichier est donnée par les indicateurs de présence d'images propres à la sous-liste ou à ses items. Lors d'une consultation, une image mentale associée à un item ou à une sous-liste est affichée à l'écran.

4.3.2.3. Spécification de la création d'une liste.

Il est utile de rappeler le sens donné à la création d'une liste. En effet, il s'agit soit de la création d'une nouvelle liste, soit de l'ajout de sous-listes dans une liste existant par ailleurs. L'algorithme suivant spécifie ce fait :



Mise_à_jour du fichier de base : si le nom de liste encodé par l'utilisateur n'existe pas dans le fichier de base "nomliste.ind", on y ajoute un enregistrement contenant le nom de la nouvelle liste, le nom des fichiers annexes, ainsi que la taille et le format.

Charger : la procédure consistant à charger un fichier en mémoire a été décrite précédemment.

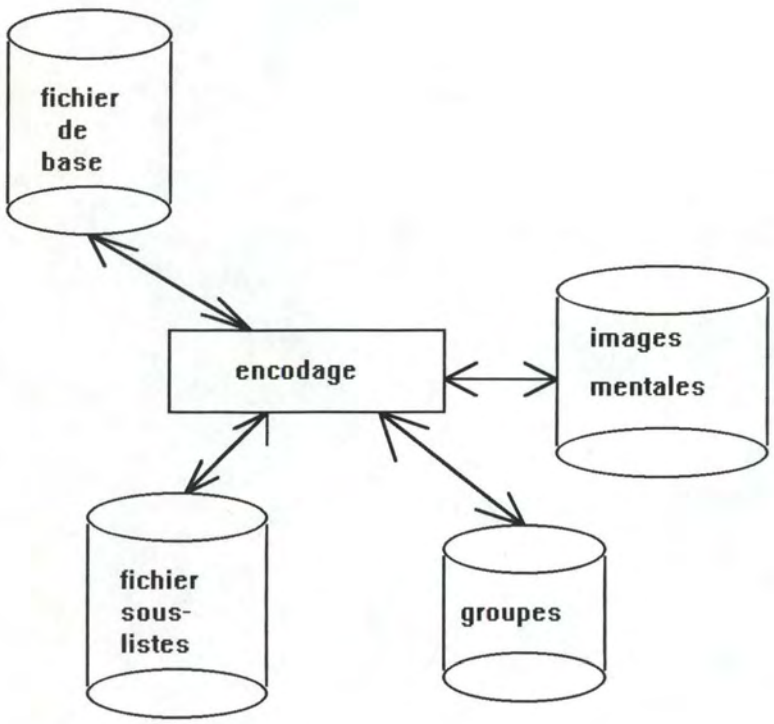
Affich_zones : il s'agit de répartir les zones d'édition à l'écran de façon à mettre en évidence la séparation entre les sous-listes simultanées à l'écran. Plus les sous-listes sont longues, moins on peut en afficher à l'écran.

Sélectionner : lorsqu'il souhaite grouper une sous-liste présente à l'écran ou lui ajouter une image mentale, l'utilisateur doit d'abord la sélectionner en cliquant sur le bouton-poussoir associé à chacune des sous-listes visibles à l'écran.

Garnir_zones : cette procédure garnit les zones d'édition avec les items des sous-listes. Pour ce faire, une table en mémoire signale les sous-listes affichées à l'écran. L'utilisateur peut parcourir la liste grâce à des fonctionnalités de défilement vers le haut et vers le bas. Lors de ces défilements, la table en mémoire est mise à jour.

4.3.2.4. Interaction entre traitements et ressources

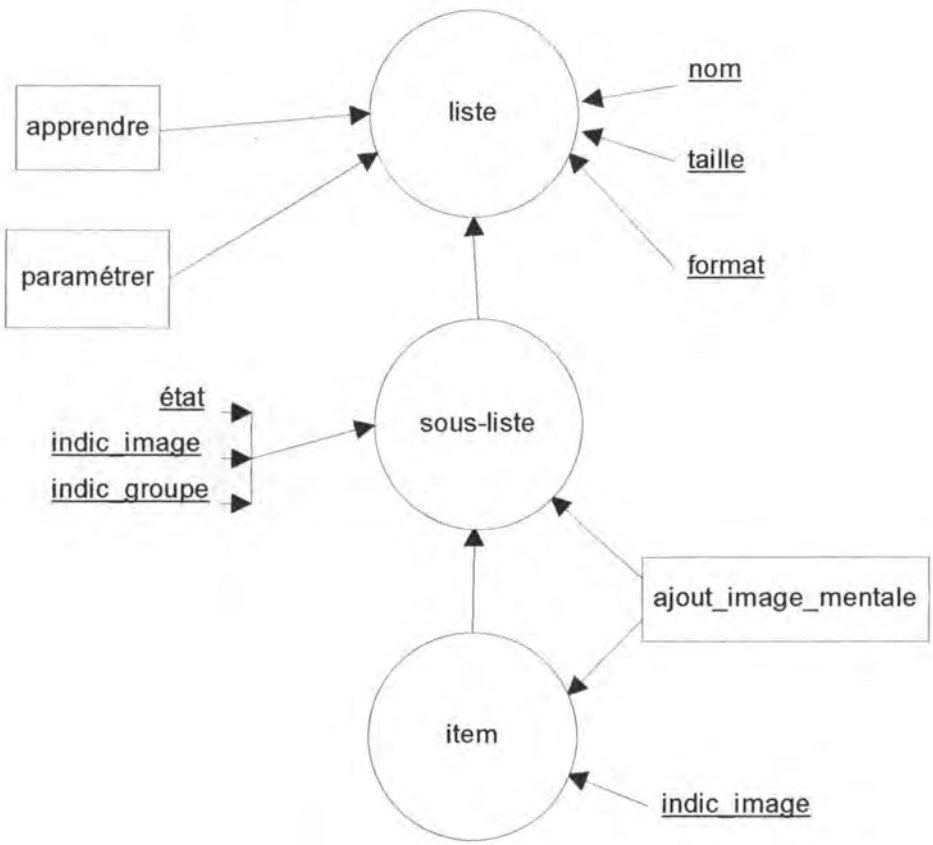
Le schéma ci-dessous décrit les flux d'informations lors de l'opération d'encodage d'une liste.



4.3.3. Apprentissage des listes

Nous allons à présent détailler les fonctionnalités liées à l'apprentissage des listes.

4.3.3.1. Représentation des objets conceptuels



4.3.3.2. Spécification de l'apprentissage d'une liste

N'est pas décrit le fonctionnement de chacun des algorithmes mis au point (cette description se trouve au chapitre 3.3.2.). Par contre, sont spécifiés les critères qu'un algorithme respecte lorsqu'il charge des sous-listes non mémorisées ou lorsqu'il indique l'entrée des sous-listes dans la mémoire à long terme de l'utilisateur.

1) Apprendre

a) les algorithmes d'apprentissage

Rappelons tout d'abord les différents types d'algorithmes d'apprentissage que nous avons conçus :

- algorithme d'étude des couples;
- algorithme pour les sous-listes de plus de deux items (taille fixe et taille variable);
- algorithme pour les sous-listes d'un seul item (listes ordonnées et désordonnées).

Lors d'un apprentissage, ces algorithmes procèdent tous de la même manière.

Précondition :

• "tab_tête_s-l" contient la partie "tête" de chaque sous-liste (état, indicateur de groupement, indicateur d'image).

• les valeurs possibles de l'indicateur d'état sont :
- état = 0 : la sous-liste doit être mémorisée,
- état = 2 : la sous-liste est mémorisée.

• "tab_bloc_s-l" est garnie avec les items de chaque sous-liste.

• seules les sous-listes dont l'indicateur d'état est à 0 sont traitées par les algorithmes d'apprentissage.

Postcondition :

L'indicateur d'état de toutes les sous-listes ayant satisfait aux conditions d'apprentissage exigées par l'algorithme passe à la valeur 2.

b) l'algorithme général de révision

Reste à spécifier le fonctionnement de l'algorithme de révision s'appliquant à tous les types de listes.

Précondition :

Seules les sous-listes dont l'indicateur d'état est à 2 sont traitées par l'algorithme de révision.

Postcondition :

Pour toutes les sous-listes révisées :

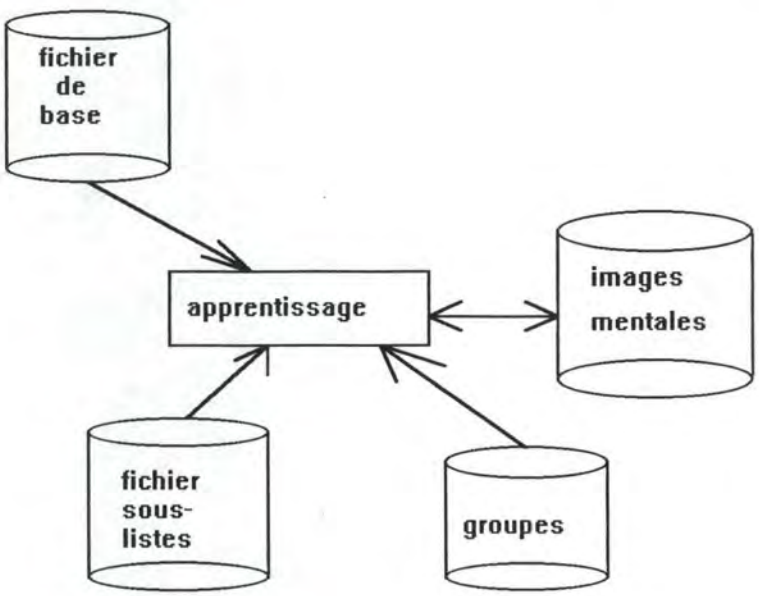
- si la sous-liste a été révisée avec succès, son indicateur d'état reste à la valeur 2.
- sinon, son indicateur d'état prend la valeur 0.

2) Paramétrer

Cette fonctionnalité permet à l'utilisateur de déterminer les différents paramètres régissant tout apprentissage. Pour rappel, ces paramètres sont :

- un indicateur sur le type et le sens du dernier apprentissage;
- le choix entre un apprentissage ou une révision, dans le mode actif ou passif;
- le choix du nombre de sous-listes par écran lors d'un apprentissage;
- le choix du ou des mots-maîtres;
- l'apprentissage avec ou sans les groupes.

4.3.3.3. Interaction entre traitements et ressources



4.3.4. Objets utilisés dans le logiciel

Nous décrivons ici les classes principales conçues afin d'implémenter les fonctionnalités liées aux listes.

4.3.4.1. Classe "Maintenance"

Cette classe contient la description des enregistrements contenus dans le fichier de base ainsi que toutes les variables nécessaires à sa gestion. Rappelons que les informations contenues dans le fichier de base sont le nom de la liste, les noms des fichiers des sous-listes, groupes et images, ainsi que la taille et le format de liste.

4.3.4.2. Classe "Sous-liste"

Dans cette classe, se trouve la description de la structure des sous-listes, c'est-à-dire la description du bloc de tête et du bloc d'item. Nous y trouvons également la description des tables "tabtetesl" et "tabblocsl" dont nous avons illustré l'utilité ci-dessus.

4.3.4.3. Classe "Objet_saisie"

Cette classe contient la description des objets de saisie apparaissant à l'écran lors d'un encodage, à savoir les zones d'édition matérialisant les items des sous-listes ainsi que les boutons-poussoirs associés à chaque sous-liste. Les zones d'édition utilisées sont construites à partir de l'objet analogue fourni par Windows, objet redéfini au niveau de la taille et des attributs (la classe "TEdition" hérite de la classe de base "TEdit" fournie par la librairie d'Object Windows).

4.3.4.4. Classe "Var_algo"

Cette classe décrit les variables et tableaux nécessaires au fonctionnement de l'algorithme des couples.

4.3.4.5. Classe "Tclassiter"

Nous trouvons ici le tableau qui gère le mécanisme des itérations, propre à l'algorithme d'apprentissage des couples. Ce tableau à deux dimensions délivre, en fonction de la fréquence d'apprentissage et du nombre de réponses correctes, le numéro de l'itération au cours de laquelle un couple donné réapparaît.

4.3.4.6. Classe "Tclassliste"

Cette classe régit les fonctionnalités d'apprentissage et d'encodage liées aux listes. Elle est constituée par héritage des cinq classes de base décrites précédemment. Elles contient également les méthodes permettant d'accéder aux objets utilisés. Nous en citerons quelques-unes à titres d'exemple :

- "up()" et "down()" : fonctions permettant d'assurer le défilement d'une liste à l'écran,
- charger_fichier() : fonction qui charge un fichier de sous-listes en mémoire,
- supprimer() : permet de supprimer une sous-liste présente à l'écran,
- algo() : cette fonction déclenche le fonctionnement de l'algorithme des couples,
- révision() : fonction représentant l'algorithme général de révision,

...

4.3.4.7. Classe "Thelp"

Cette classe est une boîte de dialogue destinée à fournir de l'aide à l'utilisateur sous forme d'un message dépendant de la barre de menu à partir de laquelle elle est appelée.

4.3.4.8. Autres classes utilisées

A chacune des boîtes de dialogue correspond une classe. Ces classes contiennent des fonctions destinées à répondre aux différentes requêtes de l'utilisateur lorsqu'il sélectionne les objets propres aux boîtes de dialogue (boutons-poussoirs, cases à cocher, boîtes de liste ...).

Toutes ces boîtes de dialogue ont deux paramètres communs :

- un pointeur de type "PTWindow" ("pt") qui fait référence à la fenêtre principale dans laquelle s'exécute l'application (cette fenêtre est construite par la classe "TFenPrinc");
- un pointeur sur la classe principale "Tclassliste", afin d'accéder aux informations qu'elle contient.

Certaines classes possèdent également une fonction intitulée "WMInitDialog". Elle a pour effet d'être exécutée automatiquement lors de la création d'une boîte de dialogue. Cela permet, par exemple, d'initialiser certaines zones d'édition ou boutons-poussoirs lors de l'apparition de la boîte à l'écran.

L'utilité de toutes ces boîtes de dialogue est commentée dans chaque fichier d'extension ".h" (voir annexes).

4.4. La découpe en modules et la structuration des menus

4.4.1. La découpe en modules

Par Philippe Vignaux

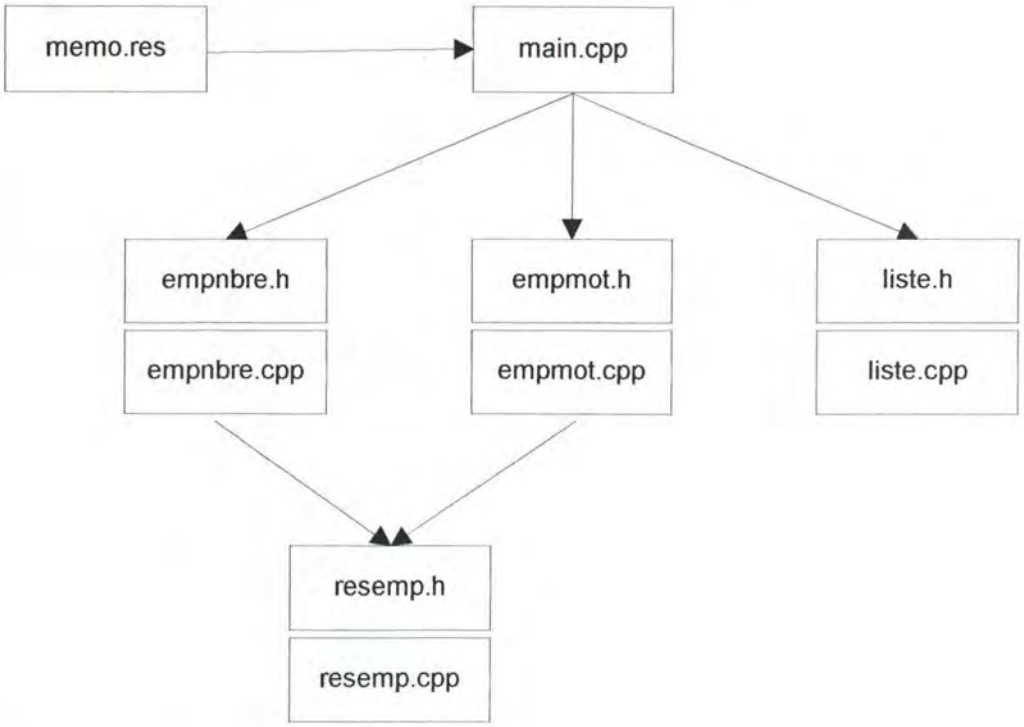
L'architecture du programme est constituée par les modules suivants :

- module de l'empan des nombres;
- module de l'empan des mots;
- module du résultat de l'empan;
- module des listes;
- module du fichier des ressources.

Un module principal assure l'interaction entre les différents modules.

En terme de fichiers, à l'exception du module des ressources et du module principal, tout autre module se décompose en deux fichiers. Un premier fichier d'extension "h" contient la déclaration des classes propres à ce module ainsi que la déclaration des méthodes appartenant à ces classes. Un second fichier d'extension "cpp" contient le code-source des méthodes déclarées dans le fichier d'extension "h" correspondant.

L'architecture schématique est donc la suivante :



Le module principal (fichier "main.cpp", voir annexe) contient la déclaration de la classe de la fenêtre principale ("TFenPrinc"). Comme les différentes barres de menus appartiennent obligatoirement à cette fenêtre, les messages de sélection relatifs à ces menus sont envoyés par Windows à celle-ci. C'est pourquoi il existe de nombreuses fonctions, dans ce fichier, destinées à répondre aux sollicitations de l'utilisateur.

Dans un même ordre d'idée, lorsqu'une boîte de dialogue est affichée à l'écran, Windows n'envoie plus les messages à la fenêtre principale de l'application mais à la classe de la boîte de dialogue en question. La boîte de dialogue est alors considérée comme la fenêtre active.

4.4.2. La structuration des menus

Par Valérie Walthéry

Cette partie décrit la hiérarchie des menus de notre logiciel. Nous ferons souvent références aux annexes afin d'illustrer notre propos par les écrans du logiciel.

Lorsque l'utilisateur exécute le programme, il accède au menu principal (voir annexes, écran 2). Ce menu comporte cinq items : « Test de l'empan », « Mode de mémorisation », « Etude de listes », « Quitter » et « Aide ».

Si l'utilisateur choisit le premier item du menu principal, le menu se déroule en deux sous-items : « Empan des nombres » et « Empan des mots » (voir annexes, écran 2). L'utilisateur choisit alors un des deux items et accède au menu de l'empan. Celui-ci est identique, qu'il s'agisse de l'empan des nombre ou de l'empan des mots (voir annexes, écran 3).

Le menu « empan » se compose de trois items : « Démarrer », « Résultats » et « Quitter ». Si l'utilisateur clique sur « Démarrer », il commence l'exercice de l'empan (voir annexes, écran 4). L'item « Résultats » permet à l'utilisateur de consulter la liste de ses résultats précédents (voir annexes, écran 5). L'item « Quitter » renvoie l'utilisateur au menu principal.

Si l'utilisateur choisit le deuxième item du menu principal, il accède au menu des modes de mémorisation. Ce menu contient les mêmes items que le menu « empan ». Si l'utilisateur clique sur l'item « Démarrer », il commence l'exécution du test sur les modes de mémorisation (lancement du traitement « Test_mode »). S'il clique sur l'item « Résultats », il voit apparaître les résultats des exercices précédents (lancement du traitement « Consulter_liste_résultats »). L'item « Quitter » renvoie l'utilisateur au menu principal.

Si l'utilisateur choisit le troisième item du menu principal, le menu se déroule en deux sous-items : « Encodage » et « Apprentissage » (voir annexes, écran 1). Quel que soit l'item choisi, une boîte de dialogue apparaît à l'écran (voir annexes, écran 7). L'utilisateur doit alors cliquer sur le nom de la liste qu'il souhaite encoder ou apprendre.

Le menu de l'encodage d'une liste comporte sept items : « Supprimer », « Image », « Groupement », « Haut », « Bas », « Quitter » et « Aide ». La fonction « Supprimer » permet d'enlever une sous-liste de la liste. Les fonctions « Image » et « Groupement » ont été définies au chapitre 3.3.1.5 (voir annexes, écrans 10 et 11). Les fonctions « Haut » et « Bas » permettent à l'utilisateur de se déplacer dans la liste. Enfin, l'item « Quitter » renvoie au menu principal et l'item « Aide » donne accès à quelques explications concernant l'encodage des listes.

Le menu de l'apprentissage d'une liste contient trois items : « Option », « Exécuter » et « Quitter » (voir annexes, écran 12). Le premier item se déroule en cinq sous-items qui correspondent aux paramètres d'apprentissage d'une liste (voir chapitre 3.3.2.1. et annexes, écran 13). Le second item démarre l'apprentissage de la liste choisie selon les paramètres définis par l'utilisateur (voir annexes, écran 14). Le dernier item renvoie au menu principal.

Le quatrième item du menu principal permet de quitter le programme. Et enfin, le dernier item donne accès à une aide.

5

CHOIX DE PROGRAMMATION ET AMELIORATIONS POTENTIELLES

Dans un premier temps, nous essayerons de justifier le choix du langage de programmation utilisé et nous argumenterons le choix des fonctions implémentées. Ensuite, nous présenterons les différentes améliorations qu'il serait intéressant d'implémenter.

5.1. Choix de programmation

Par Philippe Vignaux

5.1.1. Choix de l'environnement et du langage de programmation

Nous avons choisi d'utiliser le Borland C++ pour Windows car ce langage présente de nombreux avantages.

D'une part, la programmation sous Windows nous était jusqu'ici inconnue et nous étions donc soucieux de nous y familiariser. Nous étions également désireux d'apprendre le langage C++, langage de programmation très puissant qui, de surcroît, est un langage orienté objet. Cette technique de programmation orientée objet est une technique d'avenir et, bien que nous ayons reçu des notions théoriques, nous n'avions jamais été appelés à la maîtriser jusqu'à présent.

D'autre part, dans notre programme, l'interface joue un rôle prépondérant puisque l'utilisateur est confronté de façon permanente à un dialogue avec la machine. Ce fut pour nous une raison supplémentaire de diriger notre choix vers l'utilisation de Borland C++ puisque ce logiciel possède un outil de construction de boîtes de dialogue particulièrement performant et convivial : Workshop Ressource.

A titre indicatif, nous aurions également choisir le logiciel Visual Basic, mais il nous a semblé plus enrichissant d'utiliser l'environnement Borland.

Nous nous devons encore de mentionner qu'il nous a fallu un certain temps d'adaptation avant de maîtriser de façon acceptable ces nouvelles techniques. Ainsi, il n'est pas évident de passer d'une programmation séquentielle classique à une programmation événementielle, tout comme il n'est pas simple de maîtriser la philosophie de Windows et de ses classes d'objets hiérarchisés. Quoiqu'il en soit, nous ne regrettons nullement ces efforts tant nous avons été séduits par la puissance de Borland C++ pour Windows.

5.1.2. Choix des fonctionnalités implémentées

Il n'a pas été possible d'implémenter l'entièreté des fonctionnalités prévues. Les raisons sont, d'une part, l'abondance et la grande variété de ces dernières et, d'autre part, un manque de temps. Nous avons donc dû nous limiter et choisir de réaliser certaines fonctionnalités plutôt que d'autres. Nous allons, ici, exposer ces choix ainsi que leurs raisons.

Pour commencer, nous avons choisi de réaliser l'exercice du calcul de l'empan des nombres ainsi que l'exercice du calcul de l'empan des mots. Ces deux exercices, par leur taille limitée, présentent moins de difficultés lors de l'implémentation et nous ont permis de nous familiariser de façon plus approfondie à l'utilisation du Borland C++.

Par la suite, la réalisation de la partie traitant la mémorisation des listes a requis toute notre attention. Cette partie était pour nous la plus intéressante mais constituait également un travail de très longue haleine. C'est pourquoi nous avons choisi d'omettre l'implémentation du test déterminant les modes de mémorisation de l'utilisateur. Il n'est, de plus, pas difficile de se représenter l'apport de l'ordinateur dans un tel questionnaire.

Dans la partie traitant l'étude des listes, nous avons tout d'abord mis au point les fonctionnalités de l'encodage des listes. De fait, il est nécessaire de disposer de celles-ci avant de pouvoir mettre en oeuvre les opérations de mémorisation portant sur ces listes.

Nous avions prévu plusieurs algorithmes de mémorisation en fonction de la taille et du format des listes. Nous avons choisi d'en implémenter un de façon exhaustive plutôt que d'en aborder plusieurs de façon superficielle. Notre choix s'est porté sur l'algorithme d'étude des couples ainsi que sur toutes les fonctionnalités s'y rattachant : utilisation des images mentales, intégration des groupements de sous-listes, apprentissage d'un seul des items du couple, apprentissage des couples, algorithme de révision .

5.2. Améliorations potentielles

Par Philippe Vignaux

Dans un premier temps sont décrites les transformations que devraient subir certains modules ou certaines fonctionnalités du programme afin de remédier aux points faibles du logiciel.

Par la suite sont évoquées les fonctionnalités qui n'ont pas été réalisées à cause de leur coût d'implémentation trop élevé en regard du temps imparti pour les réaliser. D'autre part, l'utilisation du logiciel comme prototype, dans un premier temps, nous a suggéré de nouvelles idées et nous a fait découvrir de nouveaux chemins à exploiter. Enfin, il existe des fonctionnalités auxquelles nous avons pensé bien trop tardivement pour pouvoir les implémenter.

5.2.1. Modifications de fonctionnalités implémentées

5.2.1.1. Perte de place

Une première amélioration technique concerne le stockage des mots d'une liste sur le disque. Après une description sommaire du problème, nous proposerons une façon d'y remédier.

Nous avons vu précédemment qu'une liste comprenait plusieurs sous-listes, chacune de ces sous-listes étant constituée d'un certain nombre d'items pouvant varier de un à dix. Tous ces items, ainsi que les données annexes s'y rattachant (voir pages 90 et suivantes), sont stockés de façon permanente dans un fichier, à raison de un mot par emplacement fixe de vingt-cinq caractères.

Bien qu'elle soit acceptable, cette technique n'apparaît pas comme la plus performante. En effet, nous avons observé que, dans la grande majorité des cas, les mots avaient une longueur variable de huit à quinze caractères. Ainsi, pour chaque mot, environ dix caractères restent inutilisés. De plus, il est impossible d'encoder des noms dont la taille dépasse vingt-cinq caractères. Comme les fichiers ne sont jamais volumineux, la perte de place est acceptable. Dès lors, cette technique demeure utilisable. Elle ne le serait plus si les fichiers étaient plus grands.

Cependant, nous pensons que notre logiciel ne sera jamais utilisé avec des fichiers dont la taille atteindrait un voire plusieurs Mégabytes. En effet, dans le cas de l'étude de sous-listes de deux items, il serait possible d'enregistrer, sur une disquette d'une capacité de 720 Kb, quinze mille sous-listes.

Par contre, on pourrait imaginer des applications voisines de la nôtre. Ces dernières auraient pour but de mémoriser des informations différentes de celles utilisées dans nos listes et nécessiteraient de plus grands fichiers.

Dans ce cas, il conviendrait de remédier au problème décrit ci-dessus; nous proposerions d'utiliser un système qui consiste à évaluer la taille de chaque mot et à faire précéder chacun des mots dans le fichier par sa taille. Ainsi, lors de la lecture du fichier, on lit d'abord la taille qui occupe une variable entière; ensuite, on trouve le mot sur une longueur déterminée par la variable qui le précède, et ainsi de suite.

Nous n'avons pas utilisé cette technique car nous voulions disposer au plus vite des fichiers de données afin de tester nos algorithmes de mémorisation de listes. Rappelons que nous souhaitions avant tout parvenir à un programme opérationnel. C'est pourquoi nous avons privilégié une gestion simple en évitant au maximum d'implémenter des procédures qui n'étaient pas absolument nécessaires au bon fonctionnement du programme.

5.2.1.2. Allocations des données en mémoire

Lorsque l'utilisateur travaille sur une liste (encodage, apprentissage), le fichier contenant les mots de la liste en question est chargé en totalité, ce qui facilite les nombreuses opérations d'accès aux mots résultant du processus sélectionné (adjoindre à un mot un pointeur vers une association mentale, vers un groupe; supprimer le mot, notifier son assimilation à long terme...).

Il suffit alors de réclamer de l'espace-mémoire au fur et à mesure, lorsqu'on lit le fichier des données contenant les sous-listes. Cette technique est appelée allocation dynamique de mémoire.

Une amélioration du programme consisterait à réclamer des blocs de mémoire beaucoup plus grands afin d'y stocker un plus grand nombre de mots. En effet, il ne faut pas perdre de vue que, sous Windows, avant d'accéder à un bloc contenant des informations, il faut d'abord le verrouiller en mémoire. Le choix de tailles trop petites pour les blocs risque de pénaliser les performances du programme. En effet, plus les blocs sont petits, plus il en faut et plus il y a d'opérations de verrouillage et de déverrouillage en mémoire.

5.2.1.3. Plusieurs critères de groupement

Dans notre programme, il est possible de grouper certaines sous-listes en fonction d'un thème commun (voir la partie expliquant les groupements). Cette technique permet, lors d'un apprentissage, d'étudier simultanément les sous-listes appartenant à un même groupe, ce qui incite l'utilisateur à associer les mots ayant un lien entre eux.

Dans une version ultérieure du programme, il serait intéressant de pouvoir intégrer une sous-liste dans plusieurs groupements, ce qui permettrait à l'utilisateur de structurer son apprentissage en fonction de plusieurs thèmes. Certains groupements plus généraux pourraient également en englober d'autres plus particuliers (ex : le groupe animaux englobant les groupes mammifères et reptiles).

Rappelons que si l'utilisateur ne tient pas compte des groupements lors de l'étude des sous-listes, l'ordinateur propose un apprentissage dans l'ordre où celles-ci ont été encodées.

Illustrons notre propos à l'aide d'un exemple significatif. Supposons que nous traitions une liste concernant des pays du monde. Chaque sous-liste est constituée de trois items : le nom du pays, la ville capitale et le chef d'état. Cet exemple se prête bien à la constitution de deux groupements dont les critères sont :

- grouper les pays en fonction de leur continent;
- grouper les pays selon le statut du chef d'état.

Par exemple, l'utilisateur structure son apprentissage de manière à étudier d'abord toutes les monarchies d'Afrique, puis celles d'Europe etc. Il recommence le même traitement pour les républiques.

Le système est vraiment efficace à condition qu'un nombre relativement élevé de sous-listes existe. En fonction des listes à apprendre, l'utilisateur définit autant de groupements que nécessaire et choisit la structure d'apprentissage en fonction des besoins ou du moment.

Cette procédure tend à se rapprocher de la philosophie des réseaux sémantiques dans lesquels des informations situées à différents endroits du réseau sont reliées par l'intermédiaire d'une relation. Ici, dans la technique des groupements, des sous-listes sont regroupées en fonction d'un ou de plusieurs critères communs. De même, dans la technique des associations mentales, l'utilisateur attache, s'il le souhaite, une information sémantique à la sous-liste et/ou à chacun des items.

5.2.2. Description de nouvelles fonctionnalités

5.2.2.1. Analyse des erreurs

Cette fonctionnalité non implémentée n'est pas dépourvue d'intérêt puisqu'elle fournit, selon les cas, une indication sur l'erreur commise par l'utilisateur lorsqu'il étudie une liste de vocabulaire. Illustrons cette situation à l'aide d'un exemple simple.

Supposons que l'utilisateur étudie une liste de vocabulaire anglais. Parmi cette liste figurent les couples suivants :

fox : renard
rabbit : lapin
pig : cochon

Supposons encore que, lorsque l'ordinateur demande à l'utilisateur d'encoder (ou de sélectionner) le mot associé "fox", l'utilisateur se trompe et réponde le mot "lapin". Dans ce cas, l'ordinateur indique à l'utilisateur que le mot "lapin" correspond au mot anglais "rabbit". Dans la version de notre programme, l'ordinateur montre uniquement la réponse manquante, c'est-à-dire le couple 'fox-renard', mais il ne donne aucune indication sur la réponse erronée de l'utilisateur.

En cas de réponse erronée, l'ordinateur vérifie dans la liste si la réponse proposée existe dans le fichier. Cette recherche doit être la plus efficace possible puisqu'elle se déclenche à chaque erreur. C'est pourquoi nous proposons que les sous-listes de mots soient rangées dans un fichier indexé. L'index du fichier est organisé sur base de l'ordre alphabétique. Ainsi, quelle que soit la taille du fichier, il est possible de savoir, de façon immédiate, si un item donné est présent ou non dans le fichier. Notons encore que, dans le cas où les sous-listes sont des couples d'items, il faut créer deux index distincts utilisés en fonction du sens d'apprentissage choisi.

Une fonctionnalité intéressante à développer, concernant l'analyse des erreurs, est le repérage des erreurs commises plusieurs fois. En effet, lors d'un apprentissage, l'utilisateur peut commettre systématiquement la ou les mêmes erreurs sans s'en rendre

compte. Il sera donc utile que le logiciel se souvienne des erreurs commises par l'utilisateur afin de mettre ce dernier en garde contre une éventuelle répétition des erreurs.

5.2.2.2. Les fautes d'orthographe

Ce problème concerne la gestion des fautes d'orthographe lorsque l'utilisateur étudie une liste de mots et qu'il encode ceux-ci en utilisant la méthode de l'apprentissage "actif". Ce problème n'a pas reçu de solution. Précisons cependant qu'il n'est pas d'une importance capitale mais qu'il convient d'être conscient de son existence.

Qu'il s'agisse d'erreurs résultant d'une maladresse de l'utilisateur ou de fautes d'orthographe au sens classique du terme, leur répétition fréquente a pour conséquence de perturber le système d'apprentissage. En effet, s'il y a une faute d'orthographe dans un mot et si le mot encodé correspond à la réponse attendue, l'ordinateur assimile la faute d'orthographe à une erreur de connaissance du mot. Ce problème risque notamment de se manifester dans le cas de l'utilisation du logiciel par des personnes peu habituées à l'encodage sur ordinateur. Une solution consisterait à ne pas tenir compte des fautes d'orthographes puisque le logiciel a été créé afin d'aider les utilisateurs à mémoriser des items. Le fait que ces items soient bien orthographiés est évidemment un tout autre problème.

Bien qu'il existe des algorithmes de correction orthographique très efficaces, nous n'avons pas jugé intéressant d'en intégrer un dans notre logiciel car il ne s'agissait pas là d'un problème fondamental et indispensable par rapport à notre travail. La solution adoptée est simple et bénéfique pour l'utilisateur puisque ce dernier est obligé d'orthographier les mots correctement.

5.2.2.3. Développement d'un gestionnaire de fichiers

Notre logiciel utilise un grand nombre de fichiers : pour chaque liste, un fichier contient les mots et un autre les associations mentales éventuelles. Il est possible que, après avoir terminé un apprentissage, l'utilisateur veuille supprimer la liste étudiée. Nous n'avons

pas tenu compte de cette alternative. Il en va de même en ce qui concerne les groupements puisque nous n'avons pas géré la possibilité de supprimer une sous-liste du groupement auquel elle appartient.

Une autre procédure à compléter concerne la gestion de la suppression des sous-listes dans le fichier de la liste. En effet, à chaque sous-liste est associé un indicateur d'état de la sous-liste. Il signale si la liste est dans le long terme ou non. Dans le cas d'une suppression, nous ne retirons pas la sous-liste du fichier mais nous positionnons l'indicateur à '-1', ce qui signifie que la sous-liste ne sera plus chargée en mémoire lors de l'apprentissage suivant. Après un certain nombre de suppressions, il y aura donc dans le fichier une quantité d'espace perdue puisque occupée par les sous-listes effacées. Il conviendra donc de réorganiser le fichier lorsque l'on détectera que l'espace perdu est trop important.

5.2.2.4. Amélioration de l'interface

Dans notre logiciel, le rôle joué par l'interface est très important. Que ce soit lors de l'apprentissage ou lors de l'encodage, l'utilisateur interagit en permanence avec les objets fournis par Windows. Ces objets sont, entre autres, les boutons-poussoirs, les barres de menu, les zones d'édition.

A ce propos, bien que l'ergonomie soit satisfaisante pour une première version du programme, nous estimons qu'il serait intéressant de développer des objets de saisie des mots similaires à ce qui est proposé dans le tableur Excel. Dans ce logiciel, les cellules de saisie constituant une colonne adoptent toutes, par un double clic sur la souris, la taille de la cellule contenant la plus grande chaîne de caractères.

Nous devrions également prévoir la possibilité d'ajouter des titres aux colonnes constituant la liste, un ascenseur pour se déplacer plus rapidement dans la liste lors de l'encodage, ainsi que des boîtes de vérification qui témoignent pour chaque sous-liste de l'existence ou non des associations mentales.

Toutes ces améliorations auront pour conséquence d'augmenter le confort visuel et, par conséquent, le confort d'utilisation.

5.2.2.5. Utilisation du son

Il serait possible d'intégrer à notre logiciel les avantages du multimédia, notamment en ce qui concerne les documents vocaux. Il existe en effet des logiciels qui, à partir d'une chaîne ASCII, produisent le son correspondant dans le haut-parleur de la machine tout en essayant de respecter la prononciation du langage en question. Il serait alors possible à une personne chez qui le sens d'apprentissage auditif est favorisé (se référer à la théorie des modes de mémorisation) d'étudier les mots en les entendant. Cette technique serait d'un intérêt considérable pour notre programme.

Bien entendu, dans la version actuelle, si l'utilisateur estime que son sens d'apprentissage auditif est développé, il peut prononcer lui-même les mots. Il conviendrait donc d'étudier de façon plus rigoureuse ce que nous apporterait l'utilisation d'une carte son. Nous n'aborderons pas cette étude qui relève, notamment, du domaine de la psychologie.

Quoiqu'il en soit, bien que la réalisation de cette fonctionnalité ne relève pas de l'utopie, il convient de préciser que cette implémentation ne saurait avoir lieu dans un avenir proche.

5.2.2.6. Visualisation d'images

De nombreuses expériences réalisées par des chercheurs ont démontré une supériorité en mémoire des informations imagées par rapport aux informations verbales. En effet, lorsque l'on soumet à quatre groupes de personnes des listes d'actions présentées respectivement sous forme de phrases, de dessins, de photos et de films, il apparaît clairement que les représentations imagées sont mieux mémorisées que les phrases. Par ailleurs, la capacité de stockage à long terme des images semble considérable. En effet, lors d'un test, des sujets, à qui l'on avait montré plus de 2500 photos (à raison d'une photo toutes les dix secondes) ont été capables, par la suite, d'en reconnaître 90%.

Par conséquent, à l'avenir, il serait intéressant d'étudier les avantages qu'apporteraient à notre logiciel l'intégration d'un outil permettant la visualisation d'images et d'un utilitaire de dessin. Nous estimons en effet que l'utilisation d'images via l'ordinateur

pourrait s'avérer très utile lors des apprentissages à caractères éducatifs ou pédagogiques pratiqués dans les écoles.

Bien que l'utilisation du son et de l'image soient deux aspects qui s'écartent du sujet de notre mémoire, il nous a semblé opportun de signaler ce chemin intéressant à exploiter.

6

CONCLUSION

Par Valérie Walthéry et Philippe Vignaux

La mémoire humaine est un sujet assez mal maîtrisé. De nombreux psychologues ont développé des théories, parmi lesquelles celles que nous avons présentées et sur lesquelles nous nous sommes fondés. Ces théories sont parfois fort différentes. En tout cas, elles sont difficilement vérifiables scientifiquement. Un certain nombre d'hypothèses se sont ainsi présentées à nous. En choisissant la théorie du « système humain de traitement de l'information » comme base, nous avons supposé que cette théorie était vraie. Il est évident que, par la suite, une nouvelle théorie pourrait remplacer celle que nous avons choisie. Alors, il s'agirait de revoir le système d'aide à la mémorisation que nous avons développé.

D'un autre côté, l'immensité du sujet « mémoire » nous a obligés à prendre divers renseignements. En effet, le mot « mémoire » est polysémique. Nous avons alors réduit le domaine de travail et choisi une direction intéressante à nos yeux : « les listes de mots et de nombres ».

De son côté, la réalisation du logiciel nous a posé des difficultés quant à la maîtrise du nouvel environnement de travail constitué par le C++ et la programmation orientée objet. Rappelons cependant qu'au départ, nous n'avions aucune connaissance de ces outils et que la réalisation du programme de notre mémoire était pour nous l'occasion de combler cette lacune.

Malgré les fortunes diverses suscitées par ces nouvelles méthodes de travail, nous sommes satisfaits du résultat. Nous souhaitions que nos efforts débouchent sur un programme opérationnel. Tel est le cas. En effet, les tests du programme, effectués par nous ou par des tiers, ont permis de conclure que l'étude des listes de vocabulaire était facilitée et que le résultat de l'apprentissage était considérable.

Néanmoins, pour parvenir à ce résultat, nous avons dû, à notre regret, réduire la tâche imposée, dès le départ. Très vite, nous nous sommes aperçus de l'impossibilité à réaliser tout ce que nous avions prévu, non seulement par manque de temps, mais aussi en raison des difficultés décrites ci-dessus. Nous avons ainsi omis le test sur les modes de mémorisation pour nous consacrer à la réalisation de l'ensemble des fonctionnalités nécessaires à l'étude d'un type de liste, de l'encodage à la révision d'une liste de couples en passant par son apprentissage.

Enfin, le sujet de ce mémoire fut pour nous du plus grand intérêt. Conscients de n'avoir abordé qu'une partie de l'immense tâche, nous espérons néanmoins que la voie suivie, les problèmes soulevés et les éléments de réponses apportés, contribueront à une meilleure approche de l'aide à la mémorisation.

7

BIBLIOGRAPHIE

[Biblio 1] : Renaud, Jacqueline, Comment acquérir une super-mémoire, Marabout, 1988.

[Biblio 2] : Klatzky, Rita L., Human memory, structures and processes, W.H. Freeman & company, 1975.

[Biblio 3] : Lieury, Alain., La mémoire, résultats et théories, Mardaga, 1992.

[Biblio 4] : Gauquelin, Françoise, Développer sa mémoire, méthode Richeaudeau, Editions Retz, Presses Pocket, 1979.

[Biblio 5] : Weiss, D. H., Comment améliorer votre mémoire, Les guides de la réussite professionnelle, Pocket Business, 1986.

[Biblio 6] : Changeux, J-P., L'homme neuronal, Collection : « Le Temps des sciences », Fayard, 1983.

[Biblio 7] : Florès, César, La mémoire, Collection : «Que sais-je ?», Presses Universitaires de France, 1992.

[Biblio 8] : Rosenzweig, M. R., Biologie de la mémoire, Presses Universitaires de France, 1976.

[Biblio 9] : Mishkin, M. et Appenzeller, T., L'anatomie de la mémoire, Pour la science, août 1987.

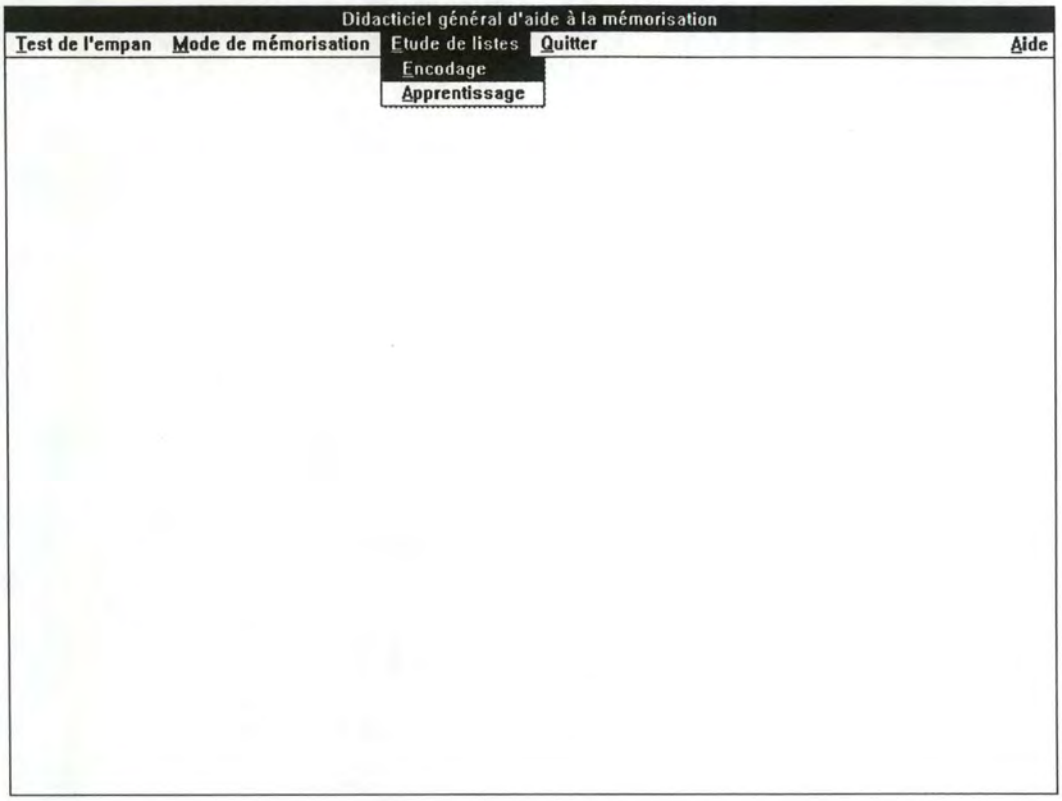
[Biblio 10] : Leblanc, Gérard, Programmation Windows en Turbo C++ et Borland C++, Eyrolles, 1992.

[Biblio 11] : Desmadril, M., La programmation sous Windows, Eyrolles.

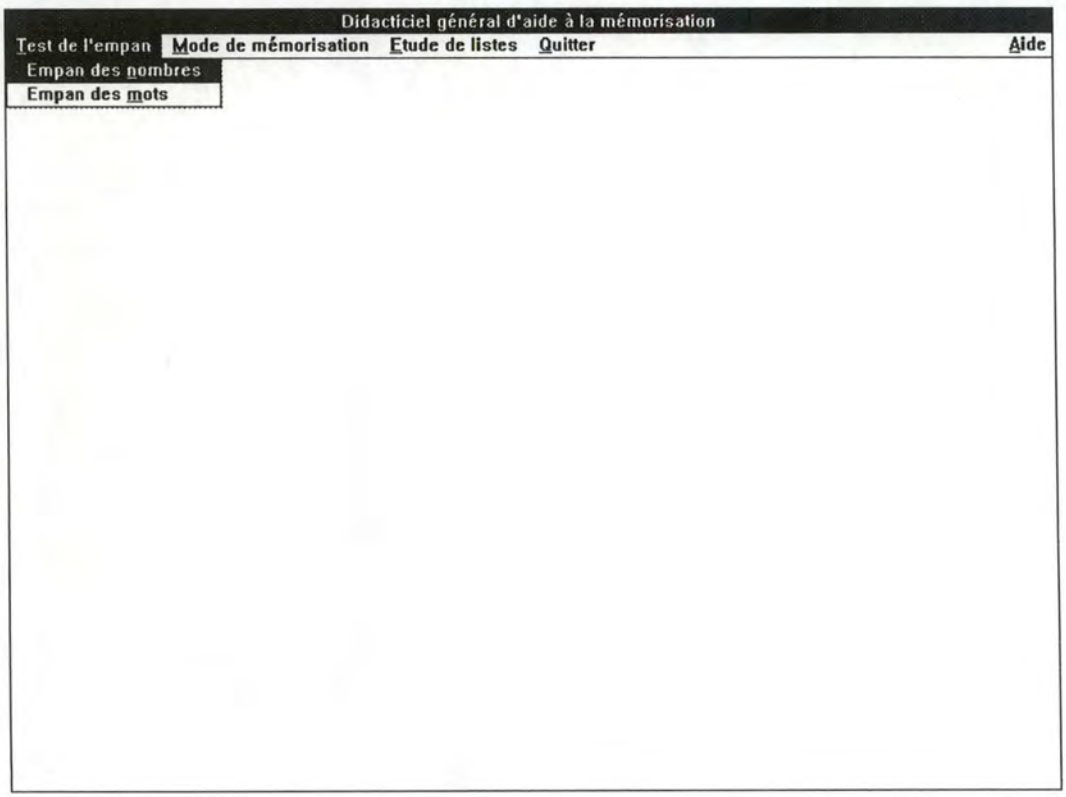
8

ANNEXES

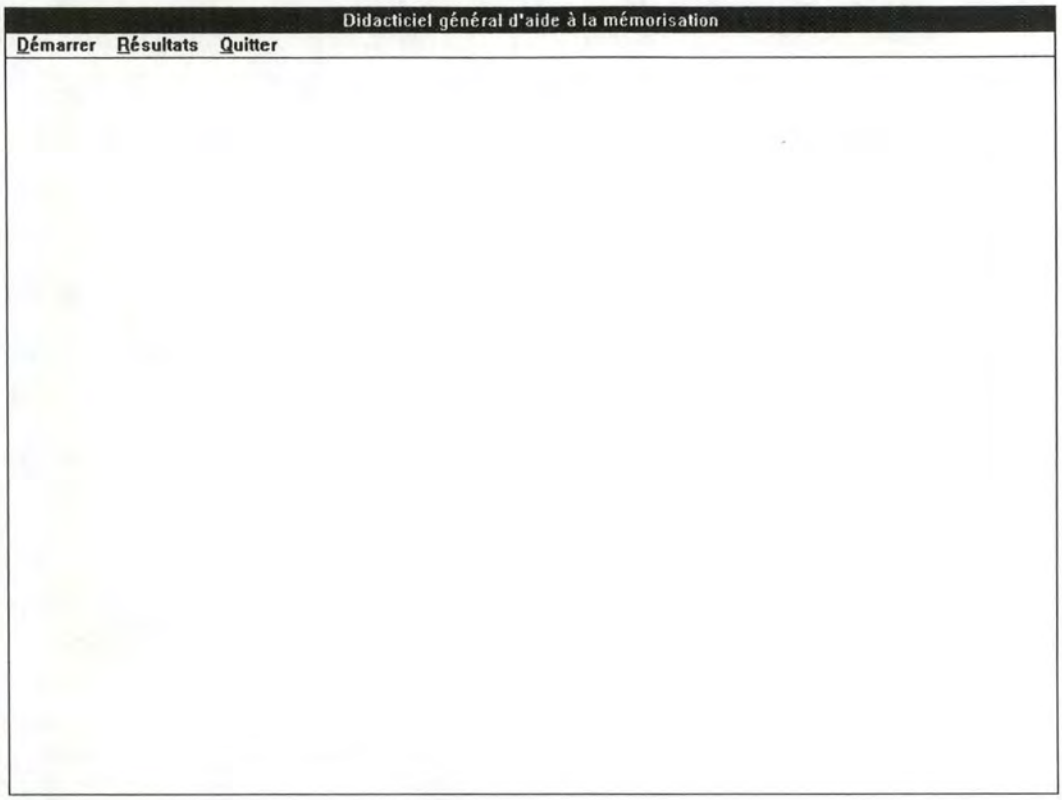
Ecrans du logiciel



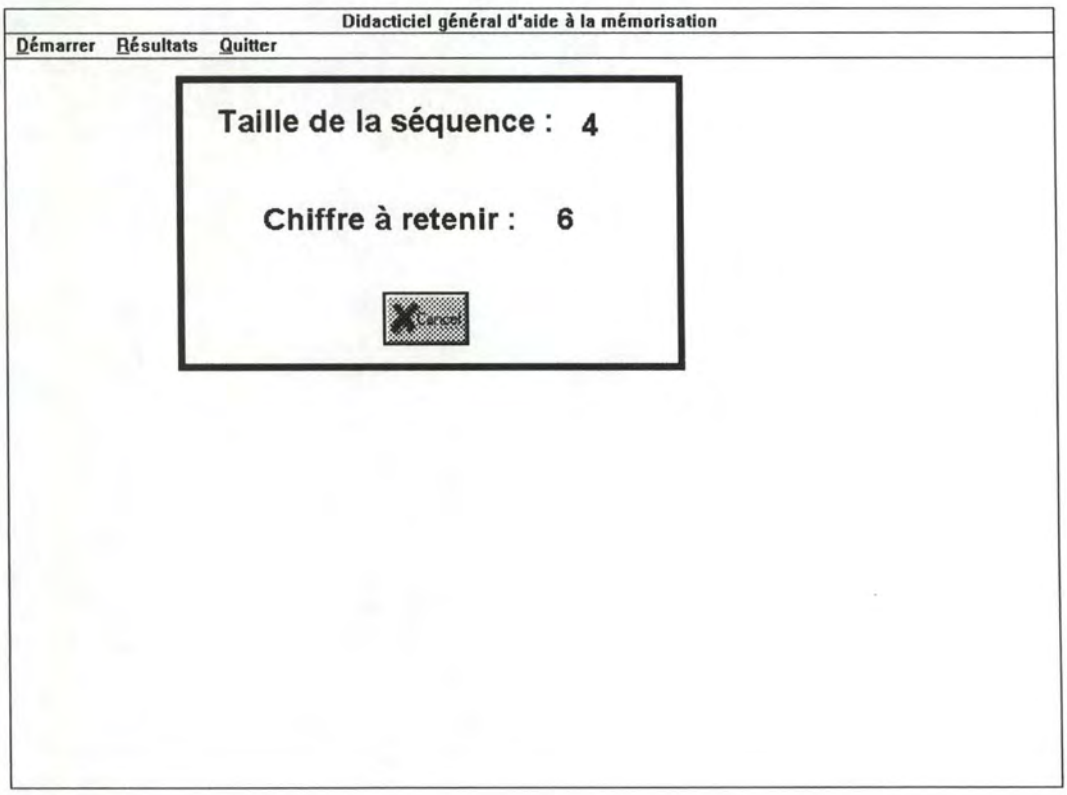
Ecran 1



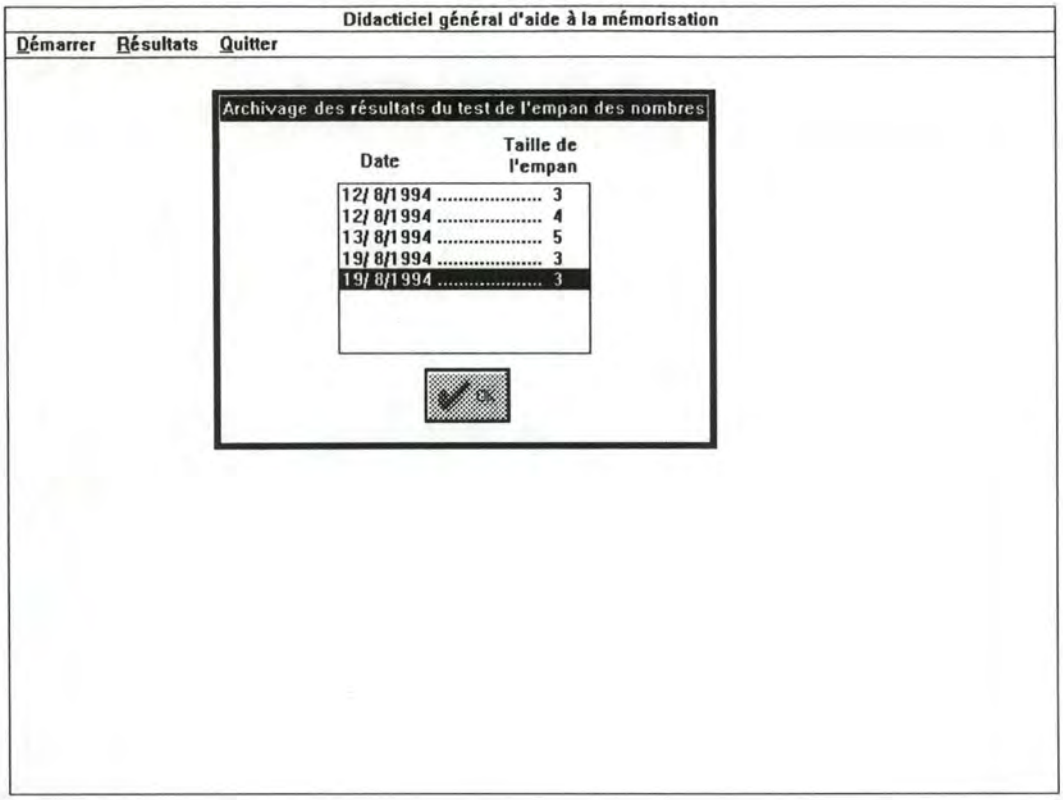
Ecran 2



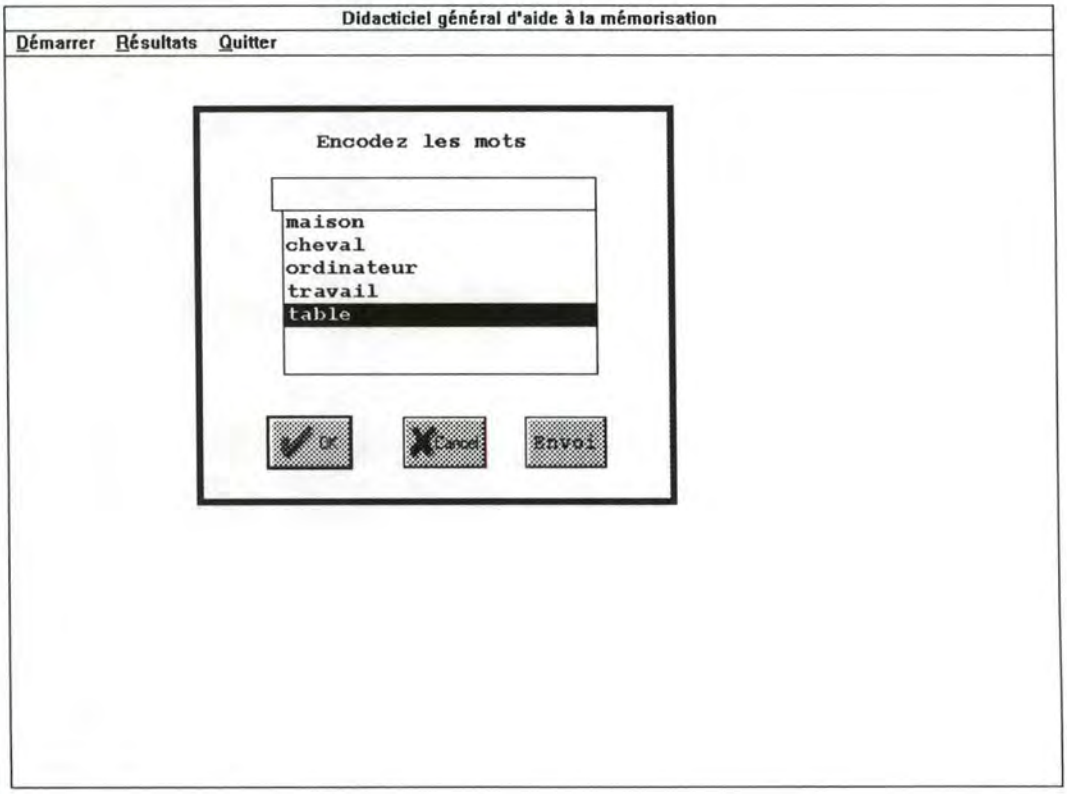
Ecran 3



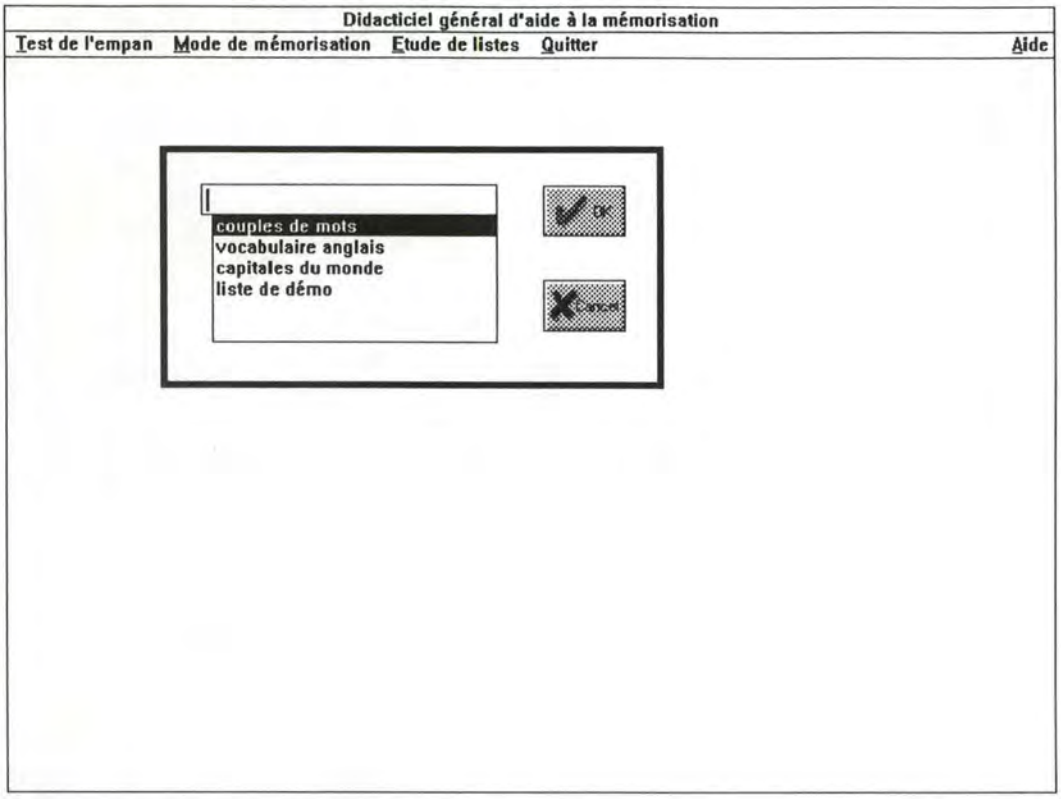
Ecran 4



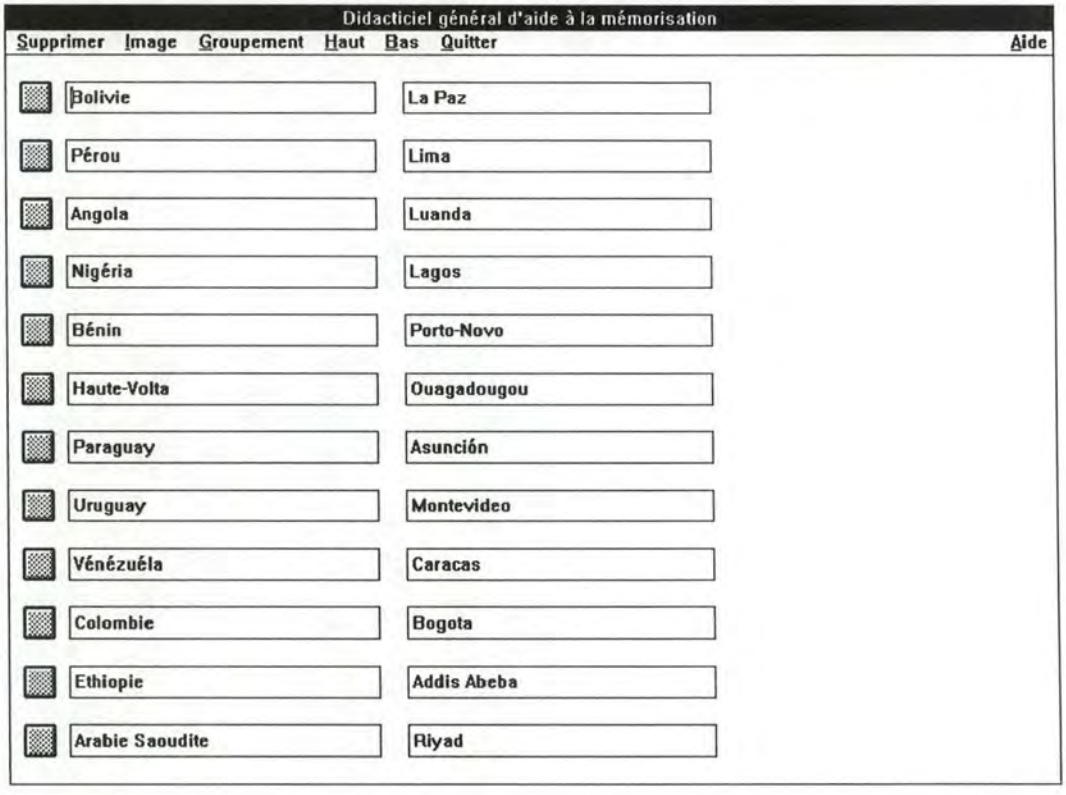
Ecran 5



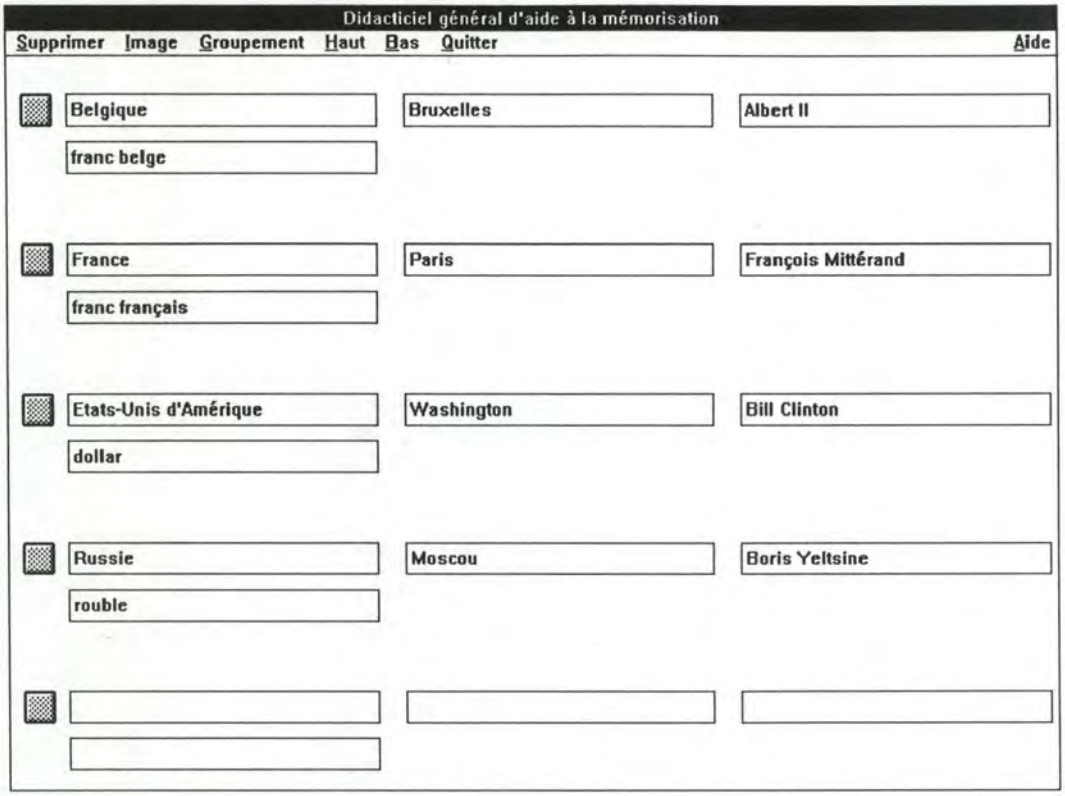
Ecran 6



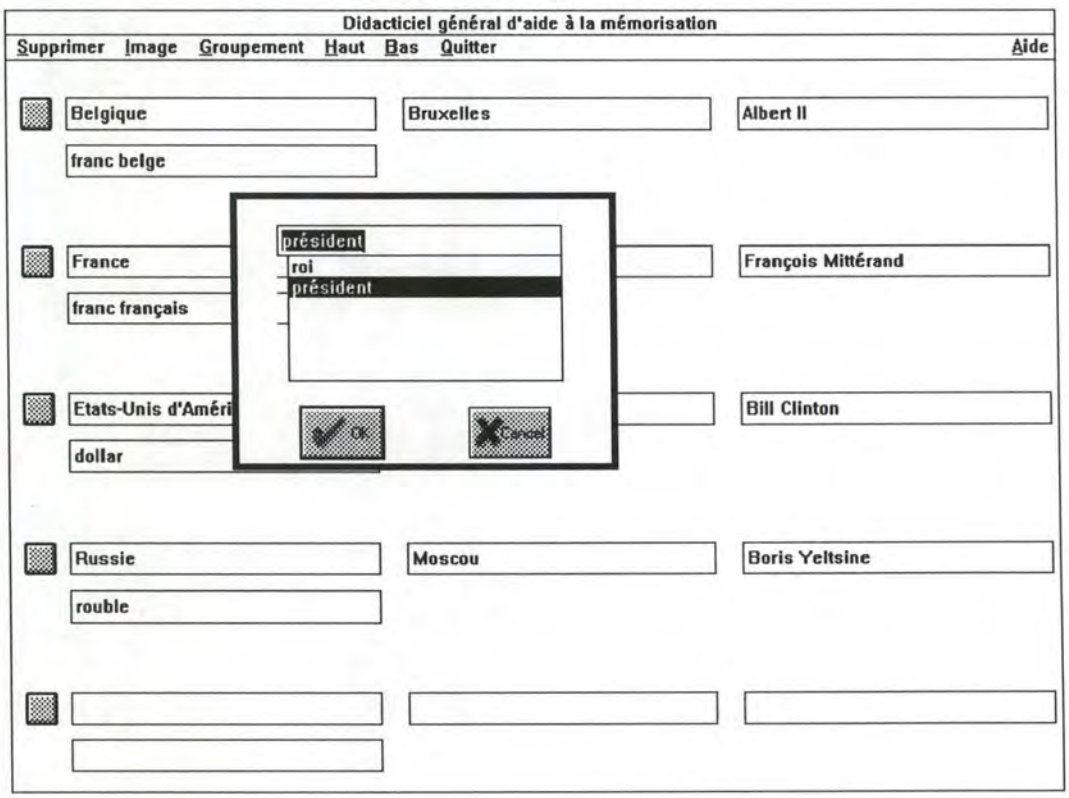
Ecran 7



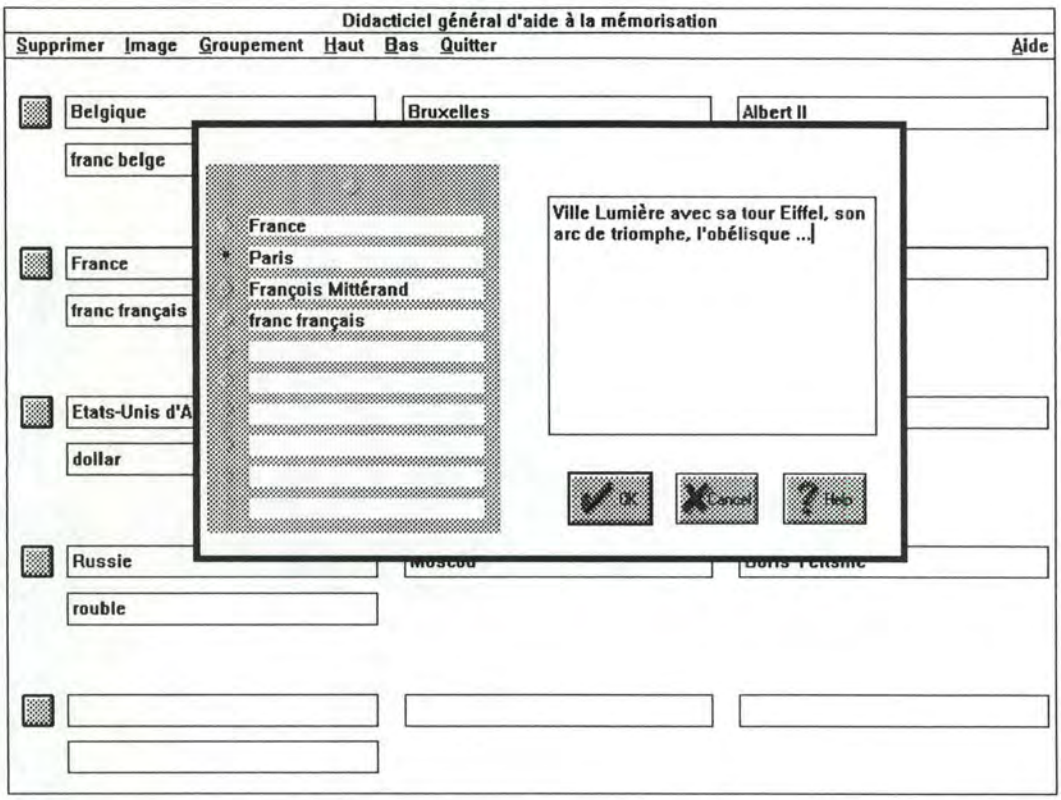
Ecran 8



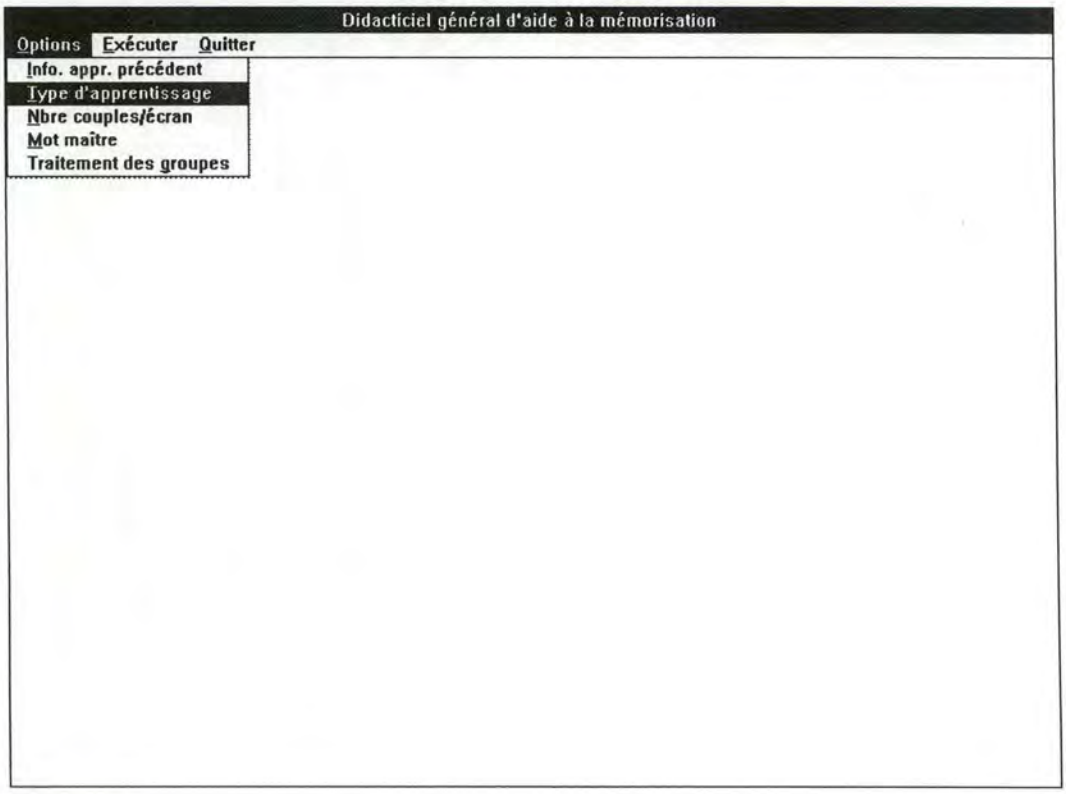
Ecran 9



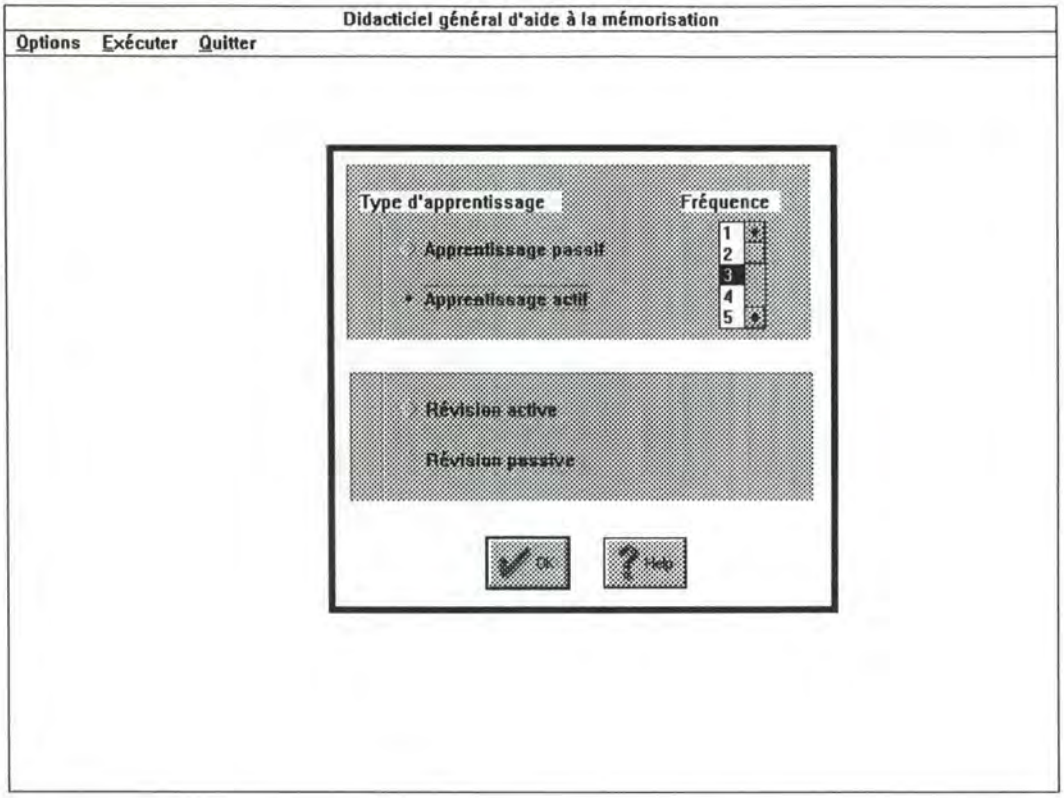
Ecran 10



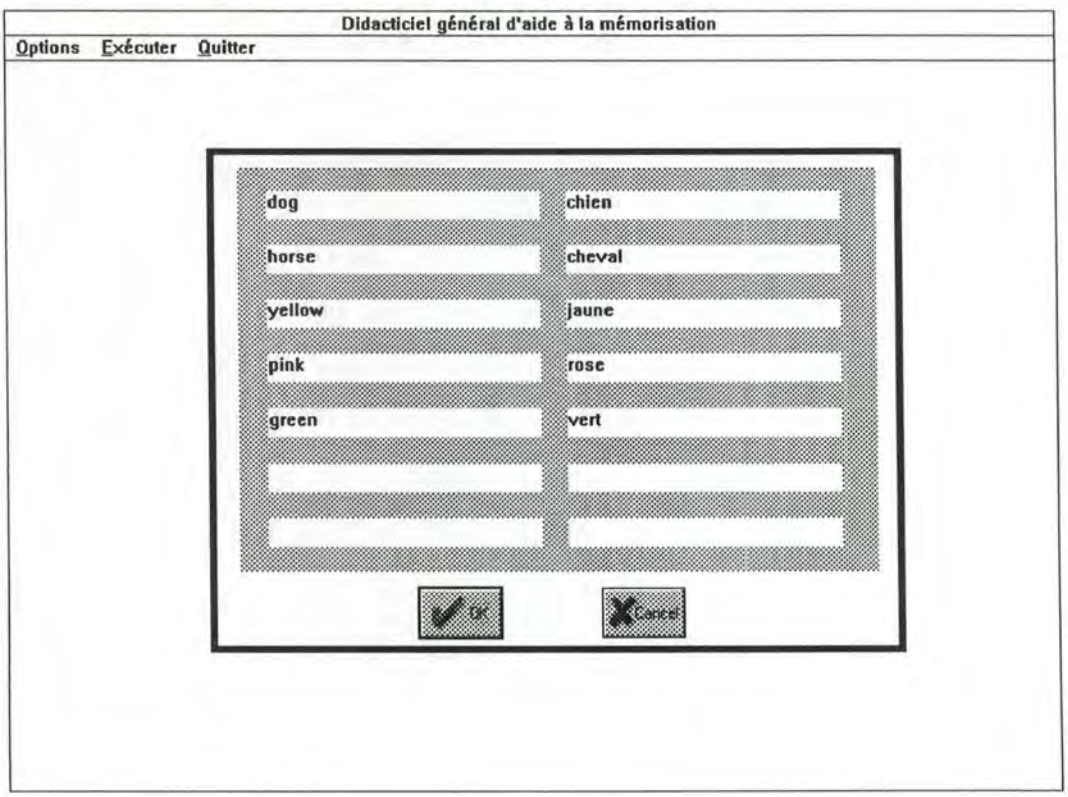
Ecran 11



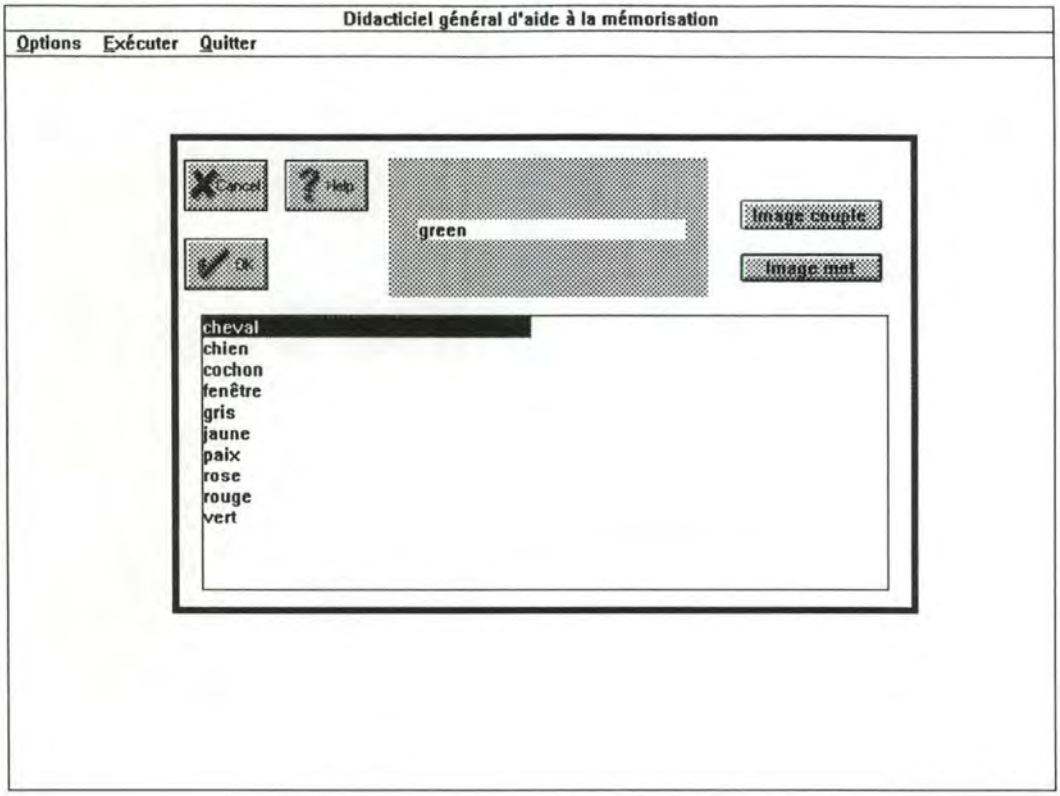
Ecran 12



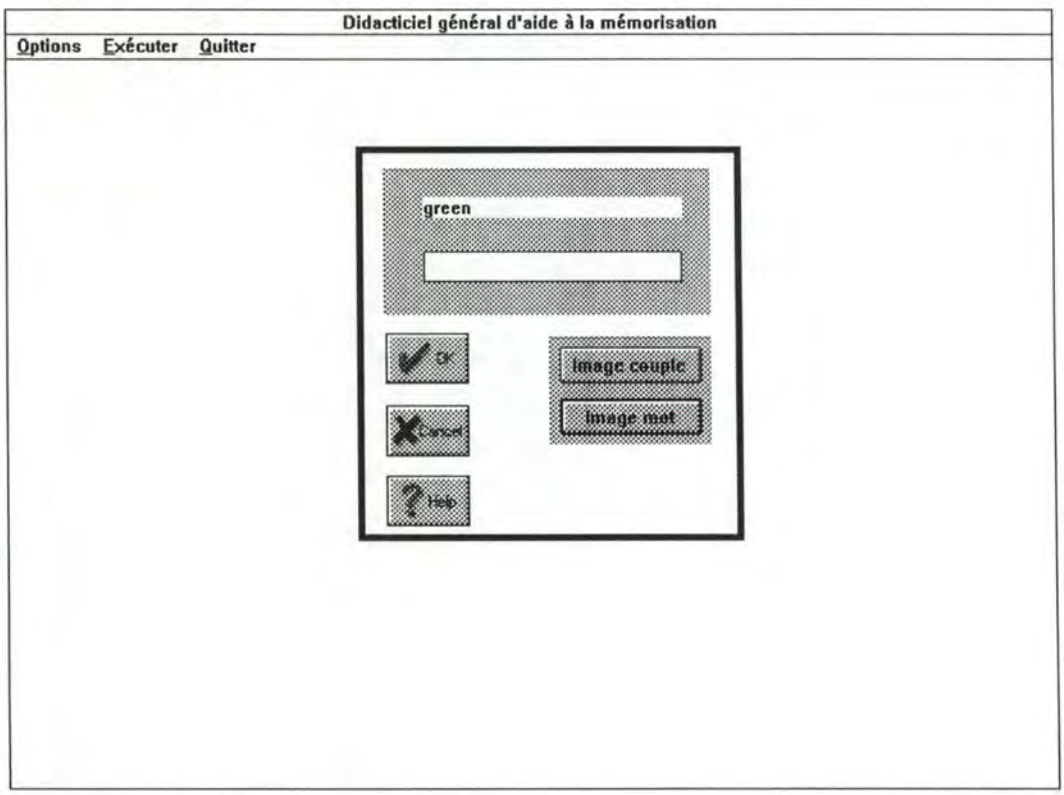
Ecran 13



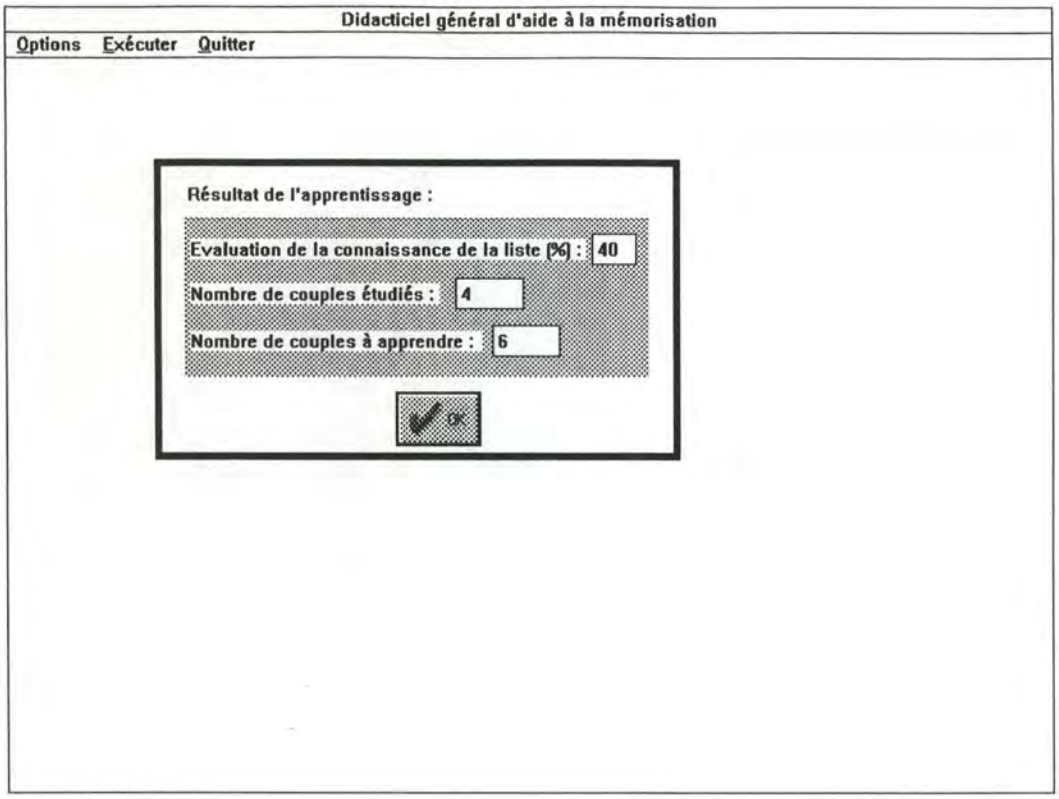
Ecran 14



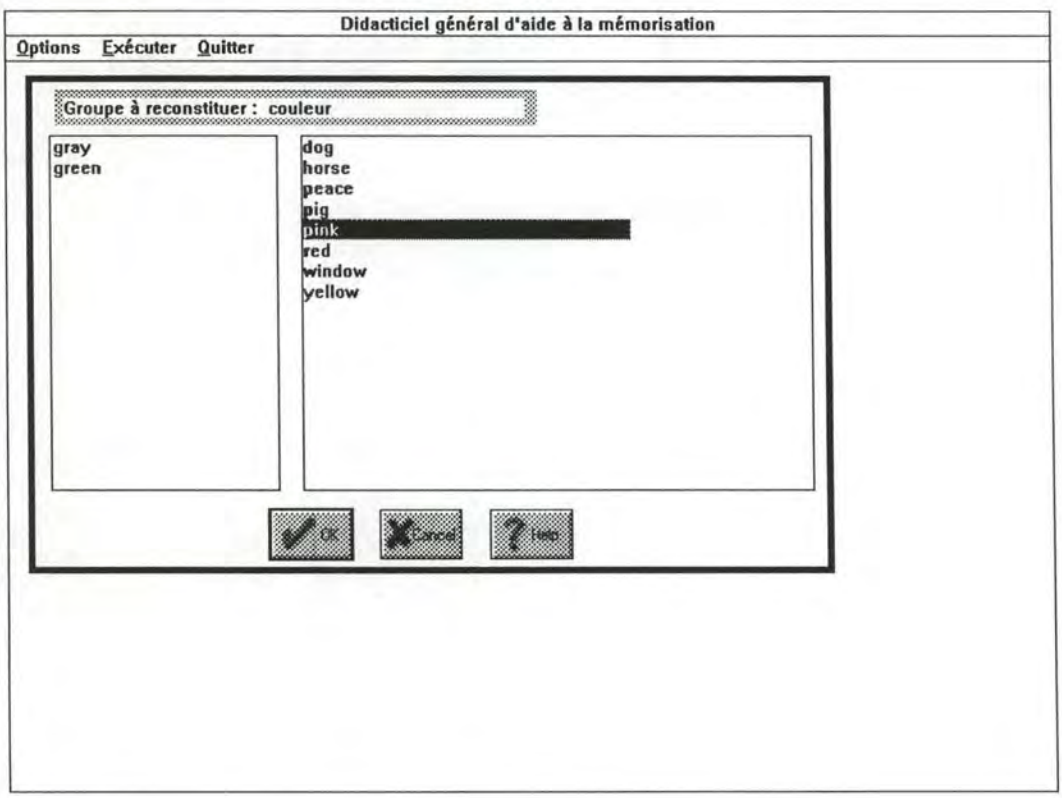
Ecran 15



Ecran 16



Ecran 17



Ecran 18

Code du logiciel

La programmation a été réalisée par Philippe Vignaux

fichier MAIN.CPP
module principal, crée les classes pour l'empan des mots, des nombres,
le traitement des listes
reçoit les appels de menus et de boutons poussoirs à l'écran

```
# ifndef WIN31
#define  WIN31
#include<owl.h>
#endif

#include "empnbre.h"
#include  "empmot.h"
#include  "resemp.h"
#include   "liste.h"

////////////////////////////////////////////////////////////////////////////////
//fenêtre principale de l'application
//reçoit les messages concernant les barres de menus, les boutons poussoirs
////////////////////////////////////////////////////////////////////////////////
class TFenPrinc : public TWindow
{
 public :
  Tempnbre*    nbre;
  Tempmot*     mot;
  Tclassliste* liste;
  Thelp* online;
  TFenPrinc(LPSTR Titre) : TWindow(NULL, Titre)
  {
   AssignMenu("menuprinc");
   Attr.X = 0;
   Attr.Y = 0;
   Attr.W = GetSystemMetrics(SM_CXSCREEN);
   Attr.H = GetSystemMetrics(SM_CYSCREEN);
   Attr.Style = WS_BORDER | WS_VISIBLE;
   nbre  = new Tempnbre();
   mot   = new Tempmot();
   liste = new Tclassliste();
  }
  void nombre(RTMessage)   = [CM_FIRST + 10];          //menu principal : empan nombres
  void mots(RTMessage)     = [CM_FIRST + 11];          //menu principal : empan mots
  void mode(RTMessage)     = [CM_FIRST + 12];          //menu principal : mode de mémorisation
  void encoder(RTMessage)  = [CM_FIRST + 13];          //menu principal : encodage listes
  void apprendre(RTMessage) = [CM_FIRST + 14];         //menu principal : étude listes
  void quitter(RTMessage)  = [CM_FIRST + 15];          //menu principal : quitter programme
  void aide(RTMessage)     = [CM_FIRST + 16];          //menu principal : obtenir aide
  void supprimer(RTMessage) = [CM_FIRST + 17];         //menu listes : supprimer sous-liste
  void image(RTMessage)    = [CM_FIRST + 18];          //menu listes : ajout image mentale
  void groupe(RTMessage)   = [CM_FIRST + 19];          //menu listes : grouper sous-listes
  void quitliste(RTMessage) = [CM_FIRST + 20];         //menu listes : retour menu principal
  void aideliste(RTMessage) = [CM_FIRST + 21];         //menu listes : obtenir aide
  void haut(RTMessage)     = [CM_FIRST + 29];          //menu listes : scrolling up
  void bas(RTMessage)      = [CM_FIRST + 30];          //menu listes : scrolling down
  void demanb(RTMessage)   = [CM_FIRST + 23];          //menu nombre : test empan nombres
  void resunb(RTMessage)   = [CM_FIRST + 24];          //menu nombre : archive test nombres
  void quitnb(RTMessage)   = [CM_FIRST + 25];          //menu nombre : retour menu principal
  void demamo(RTMessage)   = [CM_FIRST + 26];          //menu mots : test empan mots
  void resumo(RTMessage)   = [CM_FIRST + 27];          //menu mots : archive test mots
  void quitmo(RTMessage)   = [CM_FIRST + 28];          //menu mots : retour menu principal

  void trt_precedent(RTMessage) = [CM_FIRST + 35];     //param liste : info apprentissage précédent
  void trt_tapprent(RTMessage)  = [CM_FIRST + 36];     //param liste : type + fréquence
  void trt_nbcouple(RTMessage)  = [CM_FIRST + 37];     //param liste : nb couple/ecran
  void trt_motmaitre(RTMessage) = [CM_FIRST + 38];     //param liste : choix mot maître
  void trt_infogroupe(RTMessage) = [CM_FIRST + 39];    //param liste : trt des groupes.
  void trt_quiter(RTMessage)    = [CM_FIRST + 41];     //quitter liste : ret menu précédent
  void trt_executer(RTMessage)  = [CM_FIRST + 40];     // exécuter apprentissage

  void trt_bt0(RTMessage)  = [ID_FIRST + 1050];   //réponse aux clics sur les
  void trt_bt1(RTMessage)  = [ID_FIRST + 1051];   //boutons poussoirs précédant
  void trt_bt2(RTMessage)  = [ID_FIRST + 1052];   //chacune des sous-listes
  void trt_bt3(RTMessage)  = [ID_FIRST + 1053];   //maximum 12 sous-listes par
  void trt_bt4(RTMessage)  = [ID_FIRST + 1054];   //écran
```

```
  void trt_bt5(RTMessage)  = [ID_FIRST + 1055];
  void trt_bt6(RTMessage)  = [ID_FIRST + 1056];
  void trt_bt7(RTMessage)  = [ID_FIRST + 1057];
  void trt_bt8(RTMessage)  = [ID_FIRST + 1058];
  void trt_bt9(RTMessage)  = [ID_FIRST + 1059];
  void trt_bt10(RTMessage) = [ID_FIRST + 1060];
  void trt_bt11(RTMessage) = [ID_FIRST + 1061];
};

class TApp : public TApplication
{
 public :
  TApp(LPSTR Name, HANDLE hInst, HANDLE hPrevInst, LPSTR lpCmdLine, int nCmdShow)
    : TApplication(Name, hInst, hPrevInst, lpCmdLine, nCmdShow) { }
  virtual void InitMainWindow();
};

void TFenPrinc::nombre(RTMessage)
{
 AssignMenu("MENU_1");
}

void TFenPrinc::mots(RTMessage)
{
 AssignMenu("MENU_2");
}

void TFenPrinc::mode(RTMessage)
{   //non implémenté  }

void TFenPrinc::encoder(RTMessage)
{
 liste->ask_encoder(this);
}

void TFenPrinc::apprendre(RTMessage)
{
 AssignMenu("MENU_3");
 liste->appeltliste(this);
}

void TFenPrinc::quitter(RTMessage)
{
 int sortir = MessageBox(HWindow,"Etes vous sûr de vouloir sortir ?",
                                          "Quitter",MB_APPLMODAL | MB_YESNO);
 if (sortir==IDYES) PostQuitMessage(0);
}

void TFenPrinc::aide(RTMessage)
{
 GetApplication()->ExecDialog( new Thelp(0,this) );
}

void TFenPrinc::supprimer(RTMessage)
{ liste->supprimer(); }

void TFenPrinc::image(RTMessage)
{ liste->image(this); }

void TFenPrinc::groupe(RTMessage)
{ liste->groupe(this); }

void TFenPrinc::haut(RTMessage)
{ liste->up(); }

void TFenPrinc::bas(RTMessage)
{ liste->down(); }

void TFenPrinc::quitliste(RTMessage)
{
 AssignMenu("menuprinc");
 liste->sauver(1);
 liste->detruire();
```

```
}

void TFenPrinc::aideliste(RTMessage)
{
 GetApplication()->ExecDialog( new Thelp(1,this) );
}

void TFenPrinc::demanb(RTMessage)
{
 randomize();
 nbre->test_nbres(this);
}

void TFenPrinc::resunb(RTMessage)
{
 int ptr;  OFSTRUCT ofstruct;

 ptr=OpenFile("resempnb.ind",&ofstruct,OF_EXIST);
 if (ptr==-1)
   MessageBox(HWindow,"Vous n'avez pas encore effectué de test sur l'empan des nombres",
                                                                    "Attention",MB_OK);
 else
   GetApplication()->ExecDialog(new TD_Archiemp(this,1));
}

void TFenPrinc::quitnb(RTMessage)
{
 AssignMenu("menuprinc");
}

void TFenPrinc::demamo(RTMessage)
{
 randomize();
 mot->test_mots(this);
}

void TFenPrinc::resumo(RTMessage)
{
 int ptr;  OFSTRUCT ofstruct;

 ptr=OpenFile("resempmo.ind",&ofstruct,OF_EXIST);
 if (ptr==-1)
   MessageBox(HWindow,"Vous n'avez pas encore effectué de test sur l'emapn des nombres",
                                                                    "Attention",MB_OK);
 else
   GetApplication()->ExecDialog(new TD_Archiemp(this,2));
}

void TFenPrinc::quitmo(RTMessage)
{
 AssignMenu("menuprinc");
}

void TFenPrinc::trt_bt0(RTMessage)//selec = savoir sur quel bouton on a cliqué
{ MessageBeep(0);  liste->selec = 0; }

void TFenPrinc::trt_bt1(RTMessage)
{ MessageBeep(0);  liste->selec = 1; }

void TFenPrinc::trt_bt2(RTMessage)
{ MessageBeep(0);  liste->selec = 2; }

void TFenPrinc::trt_bt3(RTMessage)
{ MessageBeep(0);  liste->selec = 3; }

void TFenPrinc::trt_bt4(RTMessage)
{ MessageBeep(0);  liste->selec = 4; }

void TFenPrinc::trt_bt5(RTMessage)
{ MessageBeep(0);  liste->selec = 5; }

void TFenPrinc::trt_bt6(RTMessage)
{ MessageBeep(0);  liste->selec = 6; }
```

```
void TFenPrinc::trt_bt7(RTMessage)
{ MessageBeep(0);  liste->selec = 7; }

void TFenPrinc::trt_bt8(RTMessage)
{ MessageBeep(0);  liste->selec = 8; }

void TFenPrinc::trt_bt9(RTMessage)
{ MessageBeep(0);  liste->selec = 9; }

void TFenPrinc::trt_bt10(RTMessage)
{ MessageBeep(0);  liste->selec = 10; }

void TFenPrinc::trt_bt11(RTMessage)
{ MessageBeep(0);  liste->selec = 11; }

void Thelp::WMInitDialog(RTMessage)
{
 hwnd = GetItemHandle(457);
 ptr=OpenFile("aide.ind",&ofs,OF_READ);
 _llseek(ptr, num * sizeof texte,0);
 _lread(ptr,&texte,sizeof texte);
 SetWindowText(hwnd,texte);
 _lclose(ptr);
}

void TFenPrinc::trt_precedent(RTMessage)
{
 liste->appelprecedent(this);
}

void TFenPrinc::trt_tapprent(RTMessage)
{
 liste->appeltapprent(this);
}

void TFenPrinc::trt_nbcouple(RTMessage)
{
 liste->appelnbcouple(this);
}

void TFenPrinc::trt_motmaitre(RTMessage)
{
 liste->appelmaitre(this);
}

void TFenPrinc::trt_infogroupe(RTMessage)
{
 liste->appelgroupe(this);
}

void TFenPrinc::trt_quiter(RTMessage)
{
 AssignMenu("menuprinc");
}

void TFenPrinc::trt_executer(RTMessage)
{
 liste->appelexec(this);
}

void TApp::InitMainWindow()
{
 MainWindow = new TFenPrinc(Name);
}

int PASCAL WinMain(HANDLE hInst, HANDLE hPrevInst, LPSTR lpCmdLine, int nCmdShow)   //début d'exécution
{
 TApp App("Didacticiel général d'aide à la mémorisation", hInst,
                                          hPrevInst, lpCmdLine, nCmdShow);
 App.Run();
 return App.Status;
}
```

fichier EMPNBRE.H
contient la définition des classes pour le test de l'empan des nombres

```
#ifndef __EMPNBRE_H
#define __EMPNBRE_H

#ifndef WIN31
#define WIN31
#include<owl.h>
#endif

////////////////////////////////////////////////////////////////////
//classe de l'empan des nombres
//tab_nbre[] reçoit chaque coup les chiffres générés
//solution = proposition de l'utilisateur
////////////////////////////////////////////////////////////////////
class Tempnbre
{
 public :
   int lg_empan, arret, tab_nbre[30], pas_erreur, cpt_serie;
   char rep[20], solution[20], tamp[2];
   int i;
   Tempnbre() { };
   void test_nbres(PTWindow pt);
   void sauver_result(PTWindow pt,char *);
};

////////////////////////////////////////////////////////////////////
//boîte de dialogue affichant chacun des chiffres générés
//horloge est le timer, doit être initialisé à 1 sec.
////////////////////////////////////////////////////////////////////
class TD_Affemp : public TDialog
{
 public :
   int horloge, ind;
   char tamp[2];
   HWND hwnd;
   Tempnbre* temp;
   TD_Affemp(PTWindow pt,Tempnbre* emp) : TDialog(pt,"DIALOG_2")
   {temp=emp;}
   virtual void SetupWindow();
   virtual void Cancel(RTMessage);
   void aff(RTMessage) = [WM_TIMER];
};

////////////////////////////////////////////////////////////////////
//boîte de dialogue recevant la réponse de l'utilisateur
////////////////////////////////////////////////////////////////////
class TD_Recemp : public TDialog
{
 public :
   Tempnbre* temp;
   TD_Recemp(PTWindow pt, Tempnbre* emp) : TDialog(pt,"DIALOG_3")
   {temp=emp;}
   virtual void Ok(RTMessage);
};

#endif
```

fichier EMPNBRE.CPP

```
#include "empnbre.h"
#include<string.h>
#include<stdio.h>

void Tempnbre::test_nbres(PTWindow pt)
{
 pas_erreur=1; cpt_serie=0;
 lg_empan=3;
 arret=0;
 while((!arret) && pas_erreur)
 {
  lg_empan++;
  pas_erreur=1;
  cpt_serie=0;
  while((!arret) && pas_erreur && (cpt_serie<3) )
  {
   pas_erreur=0;
   for(i=0; i<lg_empan; i++)
   {
    tab_nbre[i] = random(10);
    itoa(tab_nbre[i],tamp,10);
    solution[i] = tamp[0];
   }
   solution[i]='\0';
   pt->GetApplication()->ExecDialog(new TD_Affemp(pt,this));
   if (!arret)
   {
          pt->GetApplication()->ExecDialog(new TD_Recemp(pt,this));
    if (strcmp(rep,solution)==0) pas_erreur=1;
    else pas_erreur=0;
    if(!pas_erreur)
          MessageBox(pt->HWindow,
          "Vous avez commis une erreur dans votre réponse",
                                "Erreur !",MB_OK);
   }
   cpt_serie++;
  }
 }
 lg_empan--;
 if(!arret)sauver_result(pt,"resempnb.ind");
}

void Tempnbre::sauver_result(PTWindow pt,char *fichname)
{
 struct {
          struct date d;
          int resu;
      }enreg;

 char comment[100];

 getdate( &(enreg.d) );
 enreg.resu = lg_empan;

 int ptr;  OFSTRUCT ofstruct;


 ptr=OpenFile(fichname,&ofstruct,OF_EXIST);
 if (ptr==-1) ptr=OpenFile(fichname,&ofstruct,OF_CREATE);
 else ptr=OpenFile(fichname,&ofstruct,OF_WRITE);
 _llseek(ptr,0L,2);
 _lwrite(ptr,(LPSTR)&enreg, sizeof enreg);
 _lclose(ptr);

 sprintf(comment,"%s %d","Résultat : ", lg_empan);
 if(lg_empan<=4)strcat(comment,". Votre empan de mémoire n'est pas très grand. Exercez-vous d'avantage.");
 else
 {
  if(lg_empan<=6)strcat(comment,". C'est un résultat acceptable. Exercez-vous encore un peu.");
  else
  {
```

```
    if(lg_empan==7)strcat(comment,". C'est un bon résultat.");
    else strcat(comment,". C'est un résultat excellent.");
   }
  }
  MessageBox(pt->HWindow,comment,"",MB_OK);
}
void TD_Affemp::SetupWindow()
{
  itoa(temp->lg_empan,tamp,10);
  hwnd = GetItemHandle(377);
  SetWindowText(hwnd,tamp);
  ind=0;
  horloge = SetTimer(HWindow,10,1000,NULL);
}

void TD_Affemp::Cancel(RTMessage)
{
  temp->arret=1;
  CloseWindow(IDCANCEL);
}

void TD_Affemp::aff(RTMessage)
{
  if (ind >= temp->lg_empan)
  {
    KillTimer(HWindow,horloge);
    CloseWindow(IDCANCEL);
  }
  else
  {
    MessageBeep(0);
    itoa( temp->tab_nbre[ind++],tamp,10);
    hwnd = GetItemHandle(375);
    SetWindowText(hwnd,tamp);
  }
}

void TD_Recemp::Ok(RTMessage)
{
  GetDlgItemText(HWindow, 376, temp->rep, 20);

  if (strcmp(temp->rep,"")!=0)
     CloseWindow(IDOK);
}
```

fichier EMPMOT.H
contient la définition des classes pour le test de l'empan des mots

```
#ifndef __EMPMOT_H
#define __EMPMOT_H

#ifndef WIN31
#define WIN31
#include<owl.h>
#endif

////////////////////////////////////////////////////////////////////////////////
//classe de l'empan de mots
//struct rmot détermine la structure d'un record pour pêcher les mots
////////////////////////////////////////////////////////////////////////////////
class Tempmot
{
 public :
  struct {
            int indice;
        char mot[20];
            }rmot;
  int lg_empan, arret, ind, pas_erreur, cpt_serie;
  int max, pos, i, ptrmot;
  char tab_emp_mot[20][20], tab_rep_mot[20][20];
  Tempmot(){};
  void test_mots(PTWindow pt);
  int correction(void);
  void sauver_result(PTWindow pt,char *);
};


//////////////////////////////////////////////////////////////////////
//boîte de dialogue affichant chacun des mots générés
//horloge est le timer à initialiser à 2 sec.
//////////////////////////////////////////////////////////////////////
class TD_Affmot : public TDialog
{
 public :
  int horloge, ind;
  char tamp[2];
  HWND hwnd;
  Tempmot* temp;
  TD_Affmot(PTWindow pt,Tempmot* emp) : TDialog(pt,"DIALOG_7")
  {temp=emp;}
  virtual void SetupWindow();
  virtual void Cancel(RTMessage);
  void aff(RTMessage) = [WM_TIMER];
};


////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue recevant les mots constituant la réponse de l'utilisateur
////////////////////////////////////////////////////////////////////////////////
class TD_Recmot : public TDialog
{
 public :
  Tempmot* temp;
  TD_Recmot(PTWindow pt,Tempmot* emp) : TDialog(pt,"DIALOG_8")
  {temp=emp;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
  void trt_envoi(RTMessage) = [ID_FIRST + 387];
};

#endif
```

fichier EMPMOT.CPP

```
#include "empmot.h"

#include<string.h>
#include<stdio.h>

void Tempmot::test_mots(PTWindow pt)
{
 pas_erreur=1; cpt_serie=0;

 ptrmot = _lopen("fichmot.ind",OF_READ);
 _llseek(ptrmot,-(long)sizeof rmot,2);
 _lread(ptrmot, (LPSTR)&rmot, sizeof(rmot));
 max=rmot.indice;  max++;

 lg_empan=3;
 arret=0;
 while((!arret) && pas_erreur)
 {
  lg_empan++;
  pas_erreur=1;
  cpt_serie=0;
  while((!arret) && pas_erreur && (cpt_serie<3) )
  {
   pas_erreur=0;
   for(i=0; i<lg_empan; i++)   // chercher 'lg_empan' mots in file
   {
            pos=random(max);
            _llseek(ptrmot,pos * sizeof rmot,0);
            _lread(ptrmot, (LPSTR)&rmot, sizeof(rmot));
            strcpy(tab_emp_mot[i],rmot.mot);
   }
   pt->GetApplication()->ExecDialog(new TD_Affmot(pt,this));
   if (!arret)
   {
    pt->GetApplication()->ExecDialog(new TD_Recmot(pt,this));
    pas_erreur = correction();
    if (!pas_erreur)
            MessageBox(pt->HWindow,"Vous avez commis une erreur dans votre réponse",
                                    "Erreur !",MB_OK);
   }
   cpt_serie++;
  }
 }
 lg_empan--;
 _lclose(ptrmot);
 if(!arret)sauver_result(pt,"resempmo.ind");
}


int Tempmot::correction(void)  //tab_emp_mot[lg_empan] tab_rep_mot[ind]
{
 int pas_erreur=1;
 if (lg_empan==ind)
 {
  for(int i=0; (i<lg_empan)&&(pas_erreur);i++)
  {
   if (strcmp(tab_emp_mot[i],tab_rep_mot[i])!=0)
           pas_erreur=0;
  }
 }
 else
  pas_erreur=0;

 return(pas_erreur);
}

void Tempmot::sauver_result(PTWindow pt,char *fichname)
{
 struct {
            struct date d;
            int resu;
```

```
            }enreg;
 char comment[100];
 int ptr;  OFSTRUCT ofstruct;
 getdate( &(enreg.d) );
 enreg.resu = lg_empan;
 ptr=OpenFile(fichname,&ofstruct,OF_EXIST);
 if (ptr==-1) ptr=OpenFile(fichname,&ofstruct,OF_CREATE);
 else ptr=OpenFile(fichname,&ofstruct,OF_WRITE);
 _llseek(ptr,0L,2);
 _lwrite(ptr,(LPSTR)&enreg, sizeof enreg);
 _lclose(ptr);
 sprintf(comment,"%s %d","Résultat : ", lg_empan);
 if(lg_empan<=4)strcat(comment,". Votre empan de mémoire n'est pas très grand. Exercez-vous d'avantage.");
 else
 {
  if(lg_empan<=6)strcat(comment,". C'est un résultat acceptable. Exercez-vous encore un peu.");
  else
  {
   if(lg_empan==7)strcat(comment,". C'est un bon résultat.");
   else strcat(comment,". C'est un résultat excellent.");
  }
 }
 MessageBox(pt->HWindow,comment,"",MB_OK);
}

void TD_Affmot::SetupWindow()
{
 itoa(temp->lg_empan,tamp,10);
 hwnd = GetItemHandle(380);
 SetWindowText(hwnd,tamp);
 ind=0;
 horloge = SetTimer(HWindow,10,2000,NULL);
}

void TD_Affmot::Cancel(RTMessage)
{
 temp->arret=1;
 CloseWindow(IDCANCEL);
}

void TD_Affmot::aff(RTMessage)
{
 if (ind>=temp->lg_empan)
 {
  KillTimer(HWindow,horloge);
  CloseWindow(IDCANCEL);
 }
 else
 {
  MessageBeep(0);
  hwnd = GetItemHandle(378);
  SetWindowText(hwnd,temp->tab_emp_mot[ind++]);
 }
}

void TD_Recmot::WMInitDialog(RTMessage)
{
 HWND hwnd;
 SendDlgItemMsg(379,CB_RESETCONTENT,0,0);
 hwnd = GetItemHandle(379);
 SetFocus(hwnd);
 temp->ind=0;
}

void TD_Recmot::Ok(RTMessage)
{
 HWND hwnd;   char buvard[20];  int pos;
 hwnd = GetItemHandle(379);
 GetWindowText(hwnd,buvard,20);
 if (strcmp(buvard,"")!=0)
 {
  pos = SendDlgItemMsg(379,CB_ADDSTRING,0,(DWORD)buvard);
  SendDlgItemMsg(379,CB_SETCURSEL,pos,0);
```

```
   strcpy(temp->tab_rep_mot[temp->ind],buvard);
   (temp->ind)++;
  }
 strcpy(buvard,"");
 SetWindowText(hwnd,buvard);
 SetFocus(hwnd);
}

void TD_Recmot::trt_envoi(RTMessage)
{
 CloseWindow(IDCANCEL);
}
```

fichier LISTE.H
contient la définition des classes, fonctions, méthodes,
boîtes de dialogue relatives à la gestion de l'encodage
et de l'apprentissage des listes

```
#ifndef __LISTE_H
#define __LISTE_H

#ifndef WIN31
#define WIN31
#include<owl.h>
#endif

#include<edit.h>
#include<button.h>


////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue affichant le texte d'aide en rapport avec la barre de menu
//ou le bouton poussoir à partir desquels l'aide est appelée
//num identifie l'endroit où l'aide est appelée, le texte édité est en rapport
////////////////////////////////////////////////////////////////////////////////////////////
class Thelp : public TDialog
{
 public :
   int num, ptr;
   char texte[1000];
   OFSTRUCT ofs;
   HWND hwnd;
   Thelp(int numbis,PTWindow pt) : TDialog(pt,"DIALOG_21")
   {num=numbis;}
   virtual void WMInitDialog(RTMessage);
};


////////////////////////////////////////////////////////////////////////////
//Redéfinition d'une zone d'édition pour saisir les items à l'écran
//la touche tab passe le focus d'une zone à la suivante
////////////////////////////////////////////////////////////////////////////
class TEdition : public TEdit
{
 public :
   HWND zsuiv;
   TEdition(PTWindowsObject pointeur, int id, LPSTR texte, int x, int y, int w)
     : TEdit(pointeur, id, texte, x, y, w, 25, 30, FALSE) { }
   void trt_tab(RTMessage) = [WM_KEYUP];
   void WMChar(RTMessage)  = [WM_CHAR];
};


////////////////////////////////////////////////////////////////////////////
//tableau pour les itérations dans l'algorithme des couples
// 2 entrées : fréquence d'apprentissage + nbre de réponse ok
////////////////////////////////////////////////////////////////////////////
class Tclassiter
{
 public :
   int incriter[6][7];
   Tclassiter()
   {
     incriter[0][0] = 2; incriter[0][1] = -1;

     incriter[1][0] = 2; incriter[1][1] =  3; incriter[1][2] = -1;

     incriter[2][0] = 2; incriter[2][1] =  2; incriter[2][2] =  5;
     incriter[2][3] = -1;

     incriter[3][0] = 2; incriter[3][1] =  2; incriter[3][2] =  3;
     incriter[3][3] = 5; incriter[3][4] = -1;

     incriter[4][0] = 2; incriter[4][1] =  2; incriter[4][2] =  3;
     incriter[4][3] = 4; incriter[4][4] =  5; incriter[4][5] = -1;
```

```
    incriter[5][0] = 2; incriter[5][1] =  2; incriter[5][2] =  3;
    incriter[5][3] = 4; incriter[5][4] =  5; incriter[5][5] =  6;
    incriter[5][6] = -1;
   }
};
////////////////////////////////////////////////////////////////////////////////
//description d'un record du fichier de base
//fichuse : fichier des s-l       imuse : fichier des images
//gpuse : fichier des groupes  deraruse : arret du dernier apprentissage
//derapuse : dernier type d'apprentissage utilisé
//local, haut, bas  utilisé pour le scrolling
////////////////////////////////////////////////////////////////////////////////
class Maintenance
{
 public :
  char fichuse[15], imuse[15], gpuse[15], titre[25];
  int foruse,tailuse,derapuse,deraruse,local,haut,bas;
  struct { int  ind_fich;
        char   intitule[25], nom_fich_sl[15];
               char nom_fich_im[15], nom_fich_gp[15];
               int  taille, format, derap, derar;
              }fiche,tampon;
  Maintenance() {}
};


////////////////////////////////////////////////////////////////////////////////
//tetesl : informations de tête d'une sous-liste
//blocsl :  "          bloc d'un item d'une sous-liste
// tabtetesl, tabblocsl  tableau pointant vers les têtes et vers les items
//it, ib  variable bornant le contenu des 2 tableaux
////////////////////////////////////////////////////////////////////////////////
class Sous_liste
{
 public :
  typedef struct {
                  int etat,ptimsl,ptgp;
                 }t_tetesl;
  t_tetesl tetesl, *ptete;
  int it;

  typedef struct {
                  char mot[25]; int ptim;
                 }t_blocsl;
  t_blocsl blocsl, *pbloc;
  int ib;

  HANDLE tabtetesl[100], tabblocsl[1000];
  Sous_liste() {}
};


////////////////////////////////////////////////////////////////////////////////
//classe réunissant les objets de saise d'un écran : zones d'édition + boutons
// poussoirs.   TEdition est la classe héritée à partir de la classe de base
// TEdit.  Tabecran indique quelles s-l sont présentes à l'écran
////////////////////////////////////////////////////////////////////////////////
class Objet_saisie
{
 public :
  int tabecran[12];
  TEdition *ze[31];
  TButton *bouton[13];
  Objet_saisie() {}
};


////////////////////////////////////////////////////////////////////////////////
//variables utilisées pour l'algorithme des couples
//tabalgo : charge les s-l pour l'algo (on travail par bloc de 20 s-l)
//tabpasssage : indique les s-l en train d'être étudiée à un moment donné
//affil[], affil2[] : totalise les bonnes réponses d'affilées dans un sens ou
```

```
//dans l'autre.  iter[] précise à quelle itération appartient une sous-liste
//tabreponse : pour chaque reponse du user -> ok ou ko
//plibre : ds tabalgo indique le nbre de s-l étudiée = le nombre de places libres
//prochar : indice de la prochaine s-l à charger par l'algo
//cpt_lt : nb de s-l dans le long terme, cpt_ap : nb de s-l à étudier
//tabgp[], indgp, gpex, apgp[] : manipulation des groupes
////////////////////////////////////////////////////////////////////////////////////////////////
class Var_algo
{
 public :
   int tabalgo[20],tabpassage[20],affil[20],affil2[20],iter[20],
           tabreponse[20],plibre,prochar,encore,cpt_lt,cpt_ap;
   int tabgp[20],indgp,gpex,rgp,apgp[50],che,encgp;
   Var_algo() {}
};


////////////////////////////////////////////////////////////////////
//classe de l'apprentissage et de l'encodage des listes
////////////////////////////////////////////////////////////////////
class Tclassliste : public Tclassiter,  public Maintenance,  public Sous_liste,
                            public Objet_saisie, public Var_algo
{
 public :
 //lg_empan : nbre de s-l par ecran lors de l'apprentissage
 //nouv : encodage dans une liste nouvelle ou dans une liste existant déjà
 //idid : indice pour le tableau où sont rangées les indices des s-l présentes à l'écran
 //nbecran : nbre de s-l présentes à l'écran en fonction de leur taille
 //apprent : indique si apprentissage actif, passif, révision
 //frequence : fréquence d'un apprentissage
 //mm : désigne le mot maître choisit dans une sous-liste
 //ptr : pointeur sur le fichier d'une liste, ptrgp : idem pour fichier des groupes d'une liste
 //apr : Révision Active Passive
  int lg_empan, nouv, idid, nbecran, apprent, frequence, mm, ptr, ptrgp, apr;
  OFSTRUCT ofstr,ofstrgpm;
  int position[100],iec,selec,rev;
  Tclassliste() : Tclassiter(), Maintenance(), Sous_liste(), Objet_saisie(), Var_algo() {}
  void ask_encoder(PTWindow pt);     //méthode pour l'encodage d'une s-l
  void appeltliste(PTWindow pt);     //paramètres
  void appelprecedent(PTWindow pt);  //  "
  void appeltapprent(PTWindow pt);   //  "
  void appelnbcouple(PTWindow pt);   //  "
  void appelmaitre(PTWindow pt);     //  "
  void appelgroupe(PTWindow pt);     //  "
  void appelexec(PTWindow pt);       //méthode lançant l'apprentissage d'une liste
  void construire(int,PTWindow);     //construction des objets de saisie à l'écran
  void detruire(void);               //destruction  "   "        "      "
  void up(void);                    //scrolling haut
  void down(void);                  //  "      bas
  void charger_fichier(int);         //les s-l vont du fichier en mémoire
  void init_scroll(int);            //garnit les zones d'edit avec les items
  void comparer(void);               //vérifie si les s-l ont étés modifiée avt scroll
  void sauver(int);                 //s-l de la mémoire vers fichier si modif !!!
  void supprimer(void);              //supprimer une s-l
  void image(PTWindow pt);           //ajout d'une image à une s-l ou un item
  void groupe(PTWindow pt);          //grouper les s-l
  int  algo(PTWindow pt);            //algo des couples
  void init_algo(void);             //init de l'algo --> sélection des premières s-l
  void correction(int);             //correction si apprent dans un seul sens
  void correctiongd(int);           // idem si apprentissage dans les 2 sens
  void envoi_lt(int);               //s-l passe dans le long terme (etat=2)
  int  revision(PTWindow pt);        //méthode de révision d'une liste
  void correv(void);                //correction des réponses du user lors d'une révision
  void rec_gp(PTWindow pt);          //traitement de révision des groupes
};


////////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue combobox pour sélectionner une s-l existant pour continuer
//l'encodage ou pour créer une nouvelle liste
////////////////////////////////////////////////////////////////////////////////////////////////
class TD_Combo : public TDialog
{
```

```
 public :
  int ptr;
  OFSTRUCT ofstruct;
  Tclassliste* ptlis;
  TD_Combo(PTWindow pt,Tclassliste* ptcli):TDialog(pt,"DIALOG_1")
  {ptlis=ptcli;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
};

////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue pour la sélection d'une liste existant en vue d'un apprentissage
////////////////////////////////////////////////////////////////////////////////////////////
class TD_Liste : public TDialog
{
 public :
  int ptrl;
  OFSTRUCT ofstrl;
  Tclassliste* ptlis;
  TD_Liste(PTWindow pt,Tclassliste* ptcli):TDialog(pt,"DIALOG_4")
  {ptlis=ptcli;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
};

////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue permettant à l'utilisateur de choisr le type d'une liste
//fixe ou variable
////////////////////////////////////////////////////////////////////////////////////////////
class TD_Tliste : public TDialog
{
 public :
  Tclassliste* ptlis;
  TD_Tliste(PTWindow pt,Tclassliste* ptcli) : TDialog(pt,"DIALOG_5")
  {ptlis=ptcli;}
  virtual void Ok(RTMessage);
  void func1(RTMessage) = [ID_FIRST + 352];
  void func2(RTMessage) = [ID_FIRST + 353];
  virtual void WMInitDialog(RTMessage);
};

////////////////////////////////////////////////////////////////////////////
//boîte de dialogue déterminant la longueur d'une liste (1->10)
////////////////////////////////////////////////////////////////////////////
class TD_Sizeliste : public TDialog
{
 public :
  Tclassliste* ptlis;
  char tamp[3];
  HWND hwnd;
  TD_Sizeliste(PTWindow pt,Tclassliste* ptcli) : TDialog(pt,"DIALOG_6")
  {ptlis=ptcli;}
  virtual void Ok(RTMessage);
  virtual void WMInitDialog(RTMessage);
};

////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue permettant de choisr le type d'apprentissage et la fréquence
////////////////////////////////////////////////////////////////////////////////////////////
class TD_Tapprent : public TDialog
{
 public :
  Tclassliste* ptlis;
  PTWindow pt;
  TD_Tapprent(PTWindow ptbi,Tclassliste* ptcli) : TDialog(ptbi,"DIALOG_11")
  {ptlis=ptcli; pt=ptbi;}
  virtual void Ok(RTMessage);
  void func1(RTMessage) = [ID_FIRST + 383];  //apr actif
  void func2(RTMessage) = [ID_FIRST + 384];  //apr passif
```

```
  void func3(RTMessage) = [ID_FIRST + 385]; //rev actif
  void func4(RTMessage) = [ID_FIRST + 387]; //rev passif
  virtual void WMInitDialog(RTMessage);
  void f_aide(RTMessage) = [ID_FIRST + 998];
};

////////////////////////////////////////////////////////////////////////
//boîte de dialogue pour le choix du ou des mots maîtres
////////////////////////////////////////////////////////////////////////
class TD_maitre : public TDialog
{
 public :
  Tclassliste* ptlis;
  TD_maitre(PTWindow pt,Tclassliste* ptbi) : TDialog(pt,"DIALOG_20")
  {ptlis=ptbi;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
};


////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue renseignant sur le type et le sens du dernier apprentissage
////////////////////////////////////////////////////////////////////////////////////////////
class TD_info : public TDialog
{
 public :
  Tclassliste* ptlis;
  HWND hwnd;
  char str[17];
  TD_info(PTWindow pt,Tclassliste* ptbi) : TDialog(pt,"DIALOG_26")
  {ptlis=ptbi;}
  virtual void WMInitDialog(RTMessage);
};


////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue pour l'ajout d'une image mentale
////////////////////////////////////////////////////////////////////////////////
class TD_Image : public TDialog
{
 public :
  int cible;
  char texte[300];
  int ptrim;
  OFSTRUCT ofstrim;
  HWND hwnd;
  Tclassliste* ptlis;
  int posintete,posinbloc,precedent;
  PTWindow pt;
  TD_Image(PTWindow ptbi,Tclassliste* ptcli,int pos) : TDialog(ptbi,"DIALOG_15")
  {ptlis=ptcli; posintete=pos; pt=ptbi;}
  virtual void WMInitDialog(RTMessage);
  void trt_image(int);
  void push0(RTMessage) = [ID_FIRST + 390]; // sélectio de l'item ou de la s-l
  void push1(RTMessage) = [ID_FIRST + 391]; // entière à laquelle on ajoute une
  void push2(RTMessage) = [ID_FIRST + 392]; // image
  void push3(RTMessage) = [ID_FIRST + 393]; // maximum 10 items
  void push4(RTMessage) = [ID_FIRST + 394];
  void push5(RTMessage) = [ID_FIRST + 395];
  void push6(RTMessage) = [ID_FIRST + 396];
  void push7(RTMessage) = [ID_FIRST + 397];
  void push8(RTMessage) = [ID_FIRST + 398];
  void push9(RTMessage) = [ID_FIRST + 399];
  void push10(RTMessage)= [ID_FIRST + 400];
  void save_prec(void);
  int ecrire(void);
  void f_aide(RTMessage) = [ID_FIRST + 998];
  virtual void Ok(RTMessage);
};


////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue pour grouper des s-l et donner un nom au groupement
```

```
////////////////////////////////////////////////////////////////////////////////
class TD_groupe : public TDialog
{
 public :
  char titre[25];
  int ptrgp;
  OFSTRUCT ofstrgp;
  HWND hwnd;
  Tclassliste* ptlis;
  TD_groupe(PTWindow pt,Tclassliste* ptcli):TDialog(pt,"DIALOG_10")
  {ptlis=ptcli;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
};


////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue pour refuser de grouper une sous-liste si elle fait déjà
// partie d'un groupement
////////////////////////////////////////////////////////////////////////////////
class TD_refgrp : public TDialog
{
 public :
  char titre[25];
  int ptrgp;
  OFSTRUCT ofstrgp;
  HWND hwnd;
  int locgp;
  Tclassliste* ptlis;
  TD_refgrp(PTWindow pt,Tclassliste* ptcli,int ptgpbis) : TDialog(pt,"DIALOG_16")
  {locgp=ptgpbis; ptlis=ptcli;}
  virtual void WMInitDialog(RTMessage);
};


////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue affichant les couples pour que l'utilisateur les mémorise
///(apprentissage actif)
////////////////////////////////////////////////////////////////////////////////
class TD_affcouple : public TDialog
{
 public :
  HWND hwnd;
  int posinbloc;
  Tclassliste* ptlis;
  TD_affcouple(PTWindow pt,Tclassliste* ptcli) : TDialog(pt,"DIALOG_12")
  {ptlis=ptcli;}
  virtual void WMInitDialog(RTMessage);
};



////////////////////////////////////////////////////////////////////////
//boîte de dialogue recevant les réponses de l'utilisateur
///(apprentissage actif)
////////////////////////////////////////////////////////////////////////
class TD_reccouple : public TDialog
{
 public :
  HWND hwnd, hwnd2;
  int posinbloc,ind,image,tabpos[20];
  char reponse[25],donne[25],autre[25];
  Tclassliste* ptlis;
  PTWindow pt;
  TD_reccouple(PTWindow ptbi,Tclassliste* ptcli) : TDialog(ptbi,"DIALOG_13")
  {ptlis=ptcli; pt=ptbi;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
  void imagecouple(RTMessage) = [ID_FIRST + 442];//appel aux images mentales lors de
  void imagemot(RTMessage) = [ID_FIRST + 443];//l'apprentissage
  void f_aide(RTMessage) = [ID_FIRST + 998];
};
```

```
////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue affichant la réponse correcte en cas d'erreur de l'utilsateur
////////////////////////////////////////////////////////////////////////////////////////////
class TD_correct : public TDialog
{
 public :
  HWND hwnd;
  TD_reccouple* ptrec;
  TD_correct(PTWindow pt,TD_reccouple* ptrecbis) : TDialog(pt,"DIALOG_17")
  {ptrec=ptrecbis;}
  virtual void WMInitDialog(RTMessage);
};


////////////////////////////////////////////////////////////////////////////
//boîte de dialogue recevant les réponses de l'utilisateur lors d'un
// apprentissage passif
////////////////////////////////////////////////////////////////////////////
class TD_recpassif : public TDialog
{
 public :
  HWND hwnd,hwnd3;
  int posinbloc,ind,image,tabpos[20],i,j,pos,sel;
  char reponse[25],donne[25],autre[25];
  Tclassliste* ptlis;
  PTWindow pt;
  TD_recpassif(PTWindow ptbi,Tclassliste* ptcli) : TDialog(ptbi,"DIALOG_14")
  {ptlis=ptcli; pt=ptbi;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
  void clicok(void);
  void trt_liste(RTMessage) = [ID_FIRST + 448];
  void imagecouple(RTMessage) = [ID_FIRST + 449];//ajout d'1 im. à la s-l lors de l'appr.
  void imagemot(RTMessage) = [ID_FIRST + 450];//ajout d'1 im. à 1 item de s-l lors de l'appr.
  void f_aide(RTMessage) = [ID_FIRST + 998];
};


////////////////////////////////////////////////////////////////////////////////////////////
//boîte de dialogue affichant la réponse correcte lors d'une erreur du user
// apprentissage passif
////////////////////////////////////////////////////////////////////////////////////////////
class TD_corpas : public TDialog
{
 public :
  HWND hwnd;
  TD_recpassif* ptrecp;
  TD_corpas(PTWindow pt,TD_recpassif* ptrecbis) : TDialog(pt,"DIALOG_18")
  {ptrecp=ptrecbis;}
  virtual void WMInitDialog(RTMessage);
};


////////////////////////////////////////////////////////////////////
//boîte de dialogue pour l'ajout d'une image mentale
////////////////////////////////////////////////////////////////////
class TD_apimage : public TDialog
{
 public :
  HWND hwndim;
  OFSTRUCT ofstrim;
  int tim,posinbloc,posintete,ptrim;
  long taille;
  int ind;
  char texte[300];
  Tclassliste* ptlis;
  TD_apimage(PTWindow pt,int tbbi,int posinbi,Tclassliste* ptclabi,int indbi) : TDialog(pt,"DIALOG_19")
  {tim=tbbi; ptlis=ptclabi; posinbloc=posinbi; ind=indbi;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
};
```

```
////////////////////////////////////////////////////////////////////////////
//boîte de dialogue affichant le résultat d'un apprentissage
////////////////////////////////////////////////////////////////////////////
class TD_result : public TDialog
{
 public :
  HWND hwnd;
  char tamp[5];
  unsigned int inter, pourcent;
  Tclassliste* ptlis;
  TD_result(PTWindow pt,Tclassliste* ptbi) : TDialog(pt,"DIALOG_22")
  {ptlis=ptbi;}
  virtual void WMInitDialog(RTMessage);
};

////////////////////////////////////////////////////////////////////////////
//boîte de dialogue affichant le résultat lors d'une révision
////////////////////////////////////////////////////////////////////////////
class TD_resurev : public TDialog
{
 public :
  HWND hwnd;
  char tamp[5];
  unsigned int inter, pourcent;
  Tclassliste* ptlis;
  TD_resurev(PTWindow pt,Tclassliste* ptbi) : TDialog(pt,"DIALOG_23")
  {ptlis=ptbi;}
  virtual void WMInitDialog(RTMessage);
};

////////////////////////////////////////////////////////////////////////////
//boîte de dialogue déterminant le nombre de couples par écran
////////////////////////////////////////////////////////////////////////////
class TD_lgempan : public TDialog
{
 public :
  char tamp[3];
  Tclassliste* ptlis;
  TD_lgempan(PTWindow pt,Tclassliste* ptbi) : TDialog(pt,"DIALOG_24")
  {ptlis=ptbi;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
};

////////////////////////////////////////////////////////////////////////////
//boîte de dialogue pour la restitution des s-l d'un groupement
////////////////////////////////////////////////////////////////////////////
class TD_recapit : public TDialog
{
 public :
  HWND hwnd;
  int posinbloc,sel,total,nbrep,faute;
  char reponse[25];
  Tclassliste* ptlis;
  PTWindow pt;
  TD_recapit(PTWindow ptbis,Tclassliste* ptbi) : TDialog(ptbis,"DIALOG_25")
  {ptlis=ptbi;pt=ptbis;}
  virtual void WMInitDialog(RTMessage);
  virtual void Ok(RTMessage);
  virtual void Cancel(RTMessage);
  void trt_selec(RTMessage) = [ID_FIRST + 465];//gestion d'1 item sélectionné lors du regroupement
  void trt_compar(void);//compare le groupe proposé par le sujet et le groupe exact
};
#endif
```

fichier LISTE.CPP

```
#include "liste.h"

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<io.h>

// pt pointeur sur FenPrinc
// this pointeur sur Tclasseliste
void Tclassliste::ask_encoder(PTWindow pt)
{
 OFSTRUCT ofs;  it=ib=0;
 int res = pt->GetApplication()->ExecDialog(new TD_Combo(pt,this));
 if (res!=IDCANCEL)
 {
  if(nouv==1)//nouvelle liste : demander format et taille
  {
   pt->GetApplication()->ExecDialog(new TD_Tliste(pt,this));
   pt->GetApplication()->ExecDialog(new TD_Sizeliste(pt,this));
   ptr=OpenFile("nomliste.ind",&ofs,OF_WRITE);
   _llseek(ptr,0L,2);
   _lwrite(ptr,(LPSTR)&fiche, sizeof fiche);
   _lclose(ptr);
  }
  else //fichier existant déjà : on le charge en memoire
  {
   ptr=OpenFile(fichuse,&ofstr,OF_READ);
   if(ptr!=-1)
          charger_fichier(1);//it donne la 1ere position libre de la tabtetesl
   else
          MessageBox(pt->HWindow,"Le fichier de la liste sélectionnée est perdu"
                                                                  ,"Erreur",MB_OK);
   _lclose(ptr);
  }
  pt->AssignMenu("menuliste");//apparition de la barre de menu des listes
  construire(tailuse,pt);//construction des zones d'édition
  init_scroll(0);//initialisation des zones d'édition
 }
}

void Tclassliste::appeltliste(PTWindow pt)
{
 OFSTRUCT ofs;
 ptr = OpenFile("nomliste.ind",&ofs,OF_EXIST);
 if (ptr!=-1)
 {
  pt->GetApplication()->ExecDialog(new TD_Liste(pt,this));
 }
 else
   MessageBox(pt->HWindow,"Aucune liste n'existe dans le didacticiel","Attention",MB_OK);
 lg_empan=5;   //initialisation par défaut des parmamètres d'apprentissage
 apprent=1;    //  "
 mm=1;         //  "
 gpex=rgp=0;   //  "
 frequence=1;  //  "

 if(derapuse==-1)//aucun apprentissage qui précède
 {
  EnableMenuItem(GetMenu(pt->HWindow),35,MF_GRAYED);//griser info sur dernier apprentissage
  DrawMenuBar(pt->HWindow);
 }
}

void Tclassliste::appelprecedent(PTWindow pt)//appel boîte de dialogue pour inf. apprent. préced.
{
 pt->GetApplication()->ExecDialog(new TD_info(pt,this));
}

void TD_info::WMInitDialog(RTMessage)//initialisation de cette boîte
{
 if(ptlis->derapuse==1)strcpy(str,"passif");
```

```
 else strcpy(str,"actif");
 hwnd=GetItemHandle(467);
 SetWindowText(hwnd,str);
 if(ptlis->deraruse==1)strcpy(str," mot de gauche");
 else
 {
  if(ptlis->deraruse==2)strcpy(str," mot de droite");
  else strcpy(str," aucun");
 }
 hwnd=GetItemHandle(468);
 SetWindowText(hwnd,str);
}

void Tclassliste::appeltapprent(PTWindow pt)//appel boîte de dialogue pour le type d'apprent.
{
 pt->GetApplication()->ExecDialog(new TD_Tapprent(pt,this));
}

void Tclassliste::appelnbcouple(PTWindow pt)//appel boîte de dialogue pour choix du nb couple/écran
{
 pt->GetApplication()->ExecDialog(new TD_Igempan(pt,this));
}

void Tclassliste::appelmaitre(PTWindow pt)//appel boîte de dialogue pour choix mot maître
{
 int pas=1;
 while(pas==1)
 {
  pt->GetApplication()->ExecDialog(new TD_maitre(pt,this));
  if(apprent!=3)
   pas=0;
  else
  {
   if(deraruse==3)
          pas=0;
   else
   {
          if((deraruse==1)&&((mm==2)||(mm==3)))MessageBox(pt->HWindow,"Vous n'avez pas étudié la liste dans ce sens. Choisissez
le premier mot comme mot maître.","Attention",MB_OK);
          if((deraruse==2)&&((mm==1)||(mm==3)))MessageBox(pt->HWindow,"Vous n'avez pas étudié la liste dans ce sens. Choisissez
le second mot comme mot maître.","Attention",MB_OK);
          if(((deraruse==1)&&(mm==1))||((deraruse==2)&&(mm==2)))pas=0;
   }
  }
 }
}

void Tclassliste::appelgroupe(PTWindow pt)//appel boîte de dialogue pour mode d'apprent. des groupes
{
 int res;
 res=MessageBox(pt->HWindow,"Souhaitez-vous un apprentissage qui tient compte des groupes existant dans votre liste
?","",MB_ICONQUESTION|MB_YESNO);
 if(res==IDNO)gpex=0;
 else
 {
  gpex=1;
  if((mm==1)||(mm==2))
  {
   res=MessageBox(pt->HWindow,"Souhaitez-vous reconstituer chaque groupe par les couples les
constituant?","",MB_ICONQUESTION|MB_YESNO);
   if(res==IDNO)rgp=0;
   else rgp=1;
  }
 }
}

void Tclassliste::appelexec(PTWindow pt)//lancement de l'apprentissage
{
 int halt=0,at=1,rt,z;  OFSTRUCT ofs;

 ib=it=0;
 ptr=OpenFile(fichuse,&ofstr,OF_READ);
 if(ptr==-1)//pas de correspondance entre le nom choisit et le fichier
```

```
{
 MessageBox(pt->HWindow,"Le fichier de la liste sélectionnée est perdu. L'apprentissage est impossible.","",MB_OK);
 goto fin;
}
cpt_lt=cpt_ap=0;
charger_fichier(2);
_lclose(ptr);

if((apprent==1)||(apprent==2))//apprentissage actif ou passif
{
 for(indgp=0;indgp<20;indgp++)
 {
  if(tabgp[indgp]!=0)at=0;
 }
 ptrgp=OpenFile(gpuse,&ofstrgpm,OF_READ);
 indgp=0;
 if((gpex==1)&&(tabgp[indgp]!=0))
 {
  strcpy(titre,"Couples non groupés");
  MessageBox(pt->HWindow,titre,"",MB_ICONINFORMATION|MB_OK);
 }
 if( (!gpex)||((gpex)&&(tabgp[indgp]!=0)) )
  halt=algo(pt);
 if(gpex==0)halt=1;
 for(indgp=1;((indgp<20)&&(!halt));indgp++)//apprentissage pour chacun des groupes existant
 {
  if(tabgp[indgp]!=0)
  {
        if(gpex==1)
   {
         _llseek(ptrgp, (sizeof titre) * (indgp-1),0);
         _lread(ptrgp,&titre,sizeof titre);//affichage du nom du groupe avant son apprentissage
         MessageBox(pt->HWindow,titre,"",MB_ICONINFORMATION|MB_OK);
   }
        for(z=0;z<50;z++)apgp[z]=0;
        halt=algo(pt);
        if((gpex==1)&&(rgp==1)&&(!halt))
        {
         encgp=1;
         while(encgp==1) rec_gp(pt);
   }
  )
 }
 _lclose(ptrgp);
}
else//révision
{
 rt=revision(pt);
}
sauver(2);
if ((apprent==1) || (apprent==2))//enregistrement des infos après un apprentissage
{
 ptr=OpenFile("nomliste.ind",&ofs,OF_READWRITE);
 _llseek(ptr, local * sizeof fiche, 0);
 _lread(ptr,(LPSTR)&fiche, sizeof fiche);
 fiche.derap = apprent;
 fiche.derar = mm;//derar=le sens d'apprent
 if(derapuse==-1)
 {
  EnableMenuItem(GetMenu(pt->HWindow),35,MF_ENABLED);
  DrawMenuBar(pt->HWindow);
 }
 derapuse=apprent;
 deraruse=mm;
 _llseek(ptr, local * sizeof fiche, 0);
 _lwrite(ptr,(LPSTR)&fiche, sizeof fiche);
 _lclose(ptr);
}
if((apprent!=3)&&(!at))pt->GetApplication()->ExecDialog(new TD_result(pt,this));
        //résultat d'un apprentissage
if((apprent==3)&&(!rt))pt->GetApplication()->ExecDialog(new TD_resurev(pt,this));
        //résultat d'une révision
if((apprent!=3)&&(at))MessageBox(pt->HWindow,"Tous les mots ont étés étudiés.  Choisissez l'option de révision.","",MB_OK);
```

```
fin:;
}


void TD_Combo::WMInitDialog(RTMessage)
{
 int val=0;   HWND hwnd;

 SendDlgItemMsg(350,CB_RESETCONTENT,0,0);
 SendDlgItemMsg(350,CB_LIMITTEXT,25,0);

 ptr = _lopen("nomliste.ind",OF_READ);

 if (ptr!=-1) // le fichier existe bien
 {
  _llseek(ptr,0L,0);
  val = _lread(ptr,(LPSTR)&(ptlis->fiche), sizeof (ptlis->fiche));
  while(val)
  {
   SendDlgItemMsg(350,CB_ADDSTRING,0,(DWORD) (ptlis->fiche).intitule );
   val = _lread(ptr,(LPSTR)&(ptlis->fiche), sizeof (ptlis->fiche));
  }
  SendDlgItemMsg(350,CB_SETCURSEL,0,0);
  hwnd=GetItemHandle(350);
  SetWindowText(hwnd,"");
  _lclose(ptr);
 }
}

void TD_Combo::Ok(RTMessage)
{
 DWORD pos, ret;  HWND hwnd;  int nb=-1; char tamp[3];

 ptlis->nouv=0;

 ptr = OpenFile("nomliste.ind",&ofstruct,OF_EXIST);
 if (ptr==-1)  ptr = OpenFile("nomliste.ind",&ofstruct,OF_CREATE);
 else ptr = OpenFile("nomliste.ind",&ofstruct,OF_READWRITE);

 hwnd = GetItemHandle(350);
 GetWindowText(hwnd,(ptlis->fiche).intitule,25);
 (ptlis->tampon).ind_fich=0;

 if (strcmp( (ptlis->fiche).intitule,"")!=0)  // le user a fait ok et a encode
 {
  pos = SendDlgItemMsg(350,CB_GETCURSEL,0,0);
  if (pos==-1)  // nouvel intitule, trouver nom_fich_sl, put to end file
  {
   ptlis->nouv=1;
   strset((ptlis->fiche).nom_fich_sl,' ');
   strset((ptlis->fiche).nom_fich_im,' ');
   strset((ptlis->fiche).nom_fich_gp,' ');
   strcpy((ptlis->fiche).nom_fich_sl,"fich");
   strcpy((ptlis->fiche).nom_fich_im,"im");
   strcpy((ptlis->fiche).nom_fich_gp,"gp");
   nb = SendDlgItemMsg(350,CB_GETCOUNT,0,0);
   if (nb>0)
   {
    nb--;
          _llseek(ptr, nb * sizeof (ptlis->tampon), 0);
          _lread(ptr,(LPSTR)&(ptlis->tampon), sizeof (ptlis->fiche));
   }
   (ptlis->fiche).ind_fich = ++((ptlis->tampon).ind_fich);
   itoa(ptlis->tampon.ind_fich,tamp,10);
   strcat((ptlis->fiche).nom_fich_sl,tamp);
   strcat((ptlis->fiche).nom_fich_sl,".ind");
   strcat((ptlis->fiche).nom_fich_im,tamp);
   strcat((ptlis->fiche).nom_fich_im,".ind");
   strcat((ptlis->fiche).nom_fich_gp,tamp);
   strcat((ptlis->fiche).nom_fich_gp,".ind");
  }
  else       // intitule reselectionne, seek en pos in file
  {
```

```
    _llseek(ptr, pos * sizeof (ptlis->fiche), 0);
    _lread(ptr,(LPSTR)&(ptlis->fiche), sizeof (ptlis->fiche));
   }
   strcpy(ptlis->fichuse,(ptlis->fiche).nom_fich_sl);
   strcpy(ptlis->imuse,  (ptlis->fiche).nom_fich_im);
   strcpy(ptlis->gpuse,  (ptlis->fiche).nom_fich_gp);
   if (ptlis->nouv != 1)
   {
    ptlis->foruse  = (ptlis->fiche).format;
    ptlis->tailuse = (ptlis->fiche).taille;
    ptlis->derapuse = (ptlis->fiche).derap;
    ptlis->deraruse = (ptlis->fiche).derar;
   }
   else//nouvelle
   {
    ptlis->derapuse = -1;
    ptlis->deraruse = -1;
    (ptlis->fiche).derap = -1;
    (ptlis->fiche).derar = -1;
   }
   CloseWindow(IDOK);
  }
  else
   SetFocus(hwnd);
  _lclose(ptr);
}

void TD_Liste::WMInitDialog(RTMessage)
{
 int val=0;

 SendDlgItemMsg(351,CB_RESETCONTENT,0,0);

 ptrl = OpenFile("nomliste.ind",&ofstrl,OF_READ);

 if (ptrl!=-1)  // le fichier existe bien
 {
  _llseek(ptrl,0L,0);
  val = _lread(ptrl,(LPSTR)&(ptlis->fiche), sizeof (ptlis->fiche));
  while(val)
  {
   SendDlgItemMsg(351,LB_ADDSTRING,0,(DWORD)((ptlis->fiche).intitule));
   val = _lread(ptrl,(LPSTR)&(ptlis->fiche), sizeof (ptlis->fiche));
  }
  SendDlgItemMsg(351,LB_SETCURSEL,0,0);
  _lclose(ptrl);
 }
}

void TD_Liste::Ok(RTMessage)
{
 int pos, ptr;

 ptr = _lopen("nomliste.ind",OF_READ);
 pos = SendDlgItemMsg(351,LB_GETCURSEL,0,0);
 _llseek(ptr, pos * sizeof (ptlis->fiche), 0);
 _lread(ptr,(LPSTR)&(ptlis->fiche), sizeof (ptlis->fiche));
 strcpy(ptlis->fichuse,(ptlis->fiche).nom_fich_sl);
 strcpy(ptlis->imuse,  (ptlis->fiche).nom_fich_im);
 strcpy(ptlis->gpuse,  (ptlis->fiche).nom_fich_gp);
 ptlis->foruse  = (ptlis->fiche).format;
 ptlis->tailuse = (ptlis->fiche).taille;
 ptlis->local = pos;
 ptlis->derapuse = (ptlis->fiche).derap;
 ptlis->deraruse = (ptlis->fiche).derar;
 CloseWindow(IDOK);
 _lclose(ptr);
}

void TD_Tliste::WMInitDialog(RTMessage)
{
 CheckDlgButton(HWindow,352,1);
}
```

```
void TD_Tliste::func1(RTMessage)
{
 int verif=IsDlgButtonChecked(HWindow,352);
 if (verif==0)  // bt 1 not checked
 {
   CheckDlgButton(HWindow,352,1);
   CheckDlgButton(HWindow,353,0);
 }
}

void TD_Tliste::func2(RTMessage)
{
 int verif=IsDlgButtonChecked(HWindow,353);
 if (verif==0)  // bt 2 not checked
 {
   CheckDlgButton(HWindow,353,1);
   CheckDlgButton(HWindow,352,0);
 }
}

void TD_Tliste::Ok(RTMessage)    // format=1 liste fixe
{                                //       2      variable
 (ptlis->fiche).format = IsDlgButtonChecked(HWindow,352) ? 1 : 2;
 ptlis->foruse = ptlis->fiche.format;
 CloseWindow(IDOK);
}

void TD_Sizeliste::WMInitDialog(RTMessage)
{
 hwnd=GetItemHandle(355);

 if ((ptlis->fiche.format)==1)
 SetWindowText(hwnd,"Choisissez le nombre de zones constituant chaque ligne de votre liste");
 else
 SetWindowText(hwnd,"Choisissez le nombre de zones maximales constituant chaque ligne de votre liste");

 SendDlgItemMsg(354,CB_RESETCONTENT,0,0);
 for(int i=1;i<11;i++)
 {
   itoa( i,tamp,10);
   SendDlgItemMsg(354,LB_ADDSTRING,0,(DWORD)tamp);
 }
 SendDlgItemMsg(354,LB_SETCURSEL,0,0);
}

void TD_Sizeliste::Ok(RTMessage)
{
 (ptlis->fiche).taille = SendDlgItemMsg(354,LB_GETCURSEL,0,0);
 ((ptlis->fiche).taille)++;
 ptlis->tailuse = ptlis->fiche.taille;
 CloseWindow(IDOK);
}

void TD_Tapprent::WMInitDialog(RTMessage)
{
 char tamp[2];

 CheckDlgButton(HWindow,383,1);

 SendDlgItemMsg(386,CB_RESETCONTENT,0,0);
 for(int i=1;i<7;i++)
 {
   itoa( i,tamp,10);
   SendDlgItemMsg(386,LB_ADDSTRING,0,(DWORD)tamp);
 }
 SendDlgItemMsg(386,LB_SETCURSEL,0,0);
}

void TD_Tapprent::func1(RTMessage)
{
 CheckDlgButton(HWindow,384,0);
```

```
 CheckDlgButton(HWindow,385,0);
 CheckDlgButton(HWindow,387,0);
}

void TD_Tapprent::func2(RTMessage)
{
 CheckDlgButton(HWindow,383,0);
 CheckDlgButton(HWindow,385,0);
 CheckDlgButton(HWindow,387,0);
}

void TD_Tapprent::func3(RTMessage)
{
 CheckDlgButton(HWindow,383,0);
 CheckDlgButton(HWindow,384,0);
 CheckDlgButton(HWindow,387,0);
}

void TD_Tapprent::func4(RTMessage)
{
 CheckDlgButton(HWindow,383,0);
 CheckDlgButton(HWindow,384,0);
 CheckDlgButton(HWindow,385,0);
}

void TD_Tapprent::f_aide(RTMessage)
{
 GetApplication()->ExecDialog( new Thelp(2,pt) );
}

void TD_Tapprent::Ok(RTMessage)
{
 int apprendre = IsDlgButtonChecked(HWindow,383);
 if (apprendre!=0)
   ptlis->apprent=1;
 else
 {
  apprendre = IsDlgButtonChecked(HWindow,384);
  if (apprendre!=0)
    ptlis->apprent=2;
  else
  {
   ptlis->apprent=3;
   apprendre = IsDlgButtonChecked(HWindow,385);
   if (apprendre!=0)
     ptlis->apr=2;
   else
     ptlis->apr=1;
  }
 }

 if ((ptlis->apprent==1)||(ptlis->apprent==2))
 {
  ptlis->frequence = SendDlgItemMsg(386,LB_GETCURSEL,0,0);
  (ptlis->frequence)++;
 }
 CloseWindow(IDOK);
}

void TD_maitre::WMInitDialog(RTMessage)
{
 CheckDlgButton(HWindow,454,TRUE);
}

void TD_maitre::Ok(RTMessage)
{
 if( IsDlgButtonChecked(HWindow,454) )
   ptlis->mm = 1;
 else
   if( IsDlgButtonChecked(HWindow,455) )
     ptlis->mm = 2;
   else
     ptlis->mm = 3;
```

```
 CloseWindow(IDOK);
}

void Tclassliste::construire(int nb,PTWindow pt)
{
 int w=GetSystemMetrics(SM_CXSCREEN); w-=30; w/=3; w-=20; //calcul taille de l'écran
 RECT rc;  GetClientRect(pt->HWindow,&rc);  // rc.bottom=hauteur réelle
 int tec=0;
 int h=GetSystemMetrics(SM_CYSCREEN)-10;
 if(h>480){h=h-480;tec=1;}
 int j, x, y, cpt, id=1000, libre, idbt=1050;

 int tab[10]  = { 12, 12,  8,  5,  5,  5,  3,  3,  3,  3};//en fct de la taille, nb de s-l/écran
 int init[10] = {-25,-25,-25,-30,-30,-30,-22,-22,-22,-22};//init en pixel pour affich 1ère s-l
 int incr[10] = { 36, 36, 36, 36, 36, 36, 42, 42, 42, 32};//incrément en pixel d'une s-l à l'autre

 nbecran=tab[nb-1];
 y=init[nb-1];  idid=0;
 h = h/tab[nb-1];

 for(int i=0;i<tab[nb-1];i++)
 {
  j=0;  libre=1;
  while(j<nb)
  {
    x=45; y+=incr[nb-1]; cpt=0;
    if (libre)
    {
          if(tec)y+=h;
          bouton[i] = new TButton(pt, idbt++,"",10,y,25,25,TRUE);//affiche un bouton
     libre=0;
    }
    while((cpt<3) && (j<nb))
    {
          ze[idid++] = new TEdition(pt, id++, "", x, y, w);//affiche une sous-liste
          pt->CreateChildren();
          x+=(w+20);
          cpt++; j++;
    }
  }
  if (nb>2) y+=20;
 }
 for(i=0;i<(idid-1);i++)
  ze[i]->zsuiv = ze[i+1]->HWindow ;
 ze[idid-1]->zsuiv = ze[idid-1]->HWindow;
}

void Tclassliste::detruire()
{
 if (idid>0)
 {
  for(int i=0;i<idid;i++)
    delete(ze[i]);//efface les zones d'édition
  for(i=0;i<nbecran;i++)
    delete(bouton[i]);//efface les boutons
 }
 idid=0;
}

void Tclassliste::charger_fichier(int mode)
{
 int reinit=0, val, i, cpt=0, sauter;
 long seekpos=0;

 if(mode==2)//mode = fct de apprent ou encod
 {if((derapuse!=-1)&&(derapuse!=apprent)&&(apprent!=3))reinit=1;}

 if((deraruse!=3)&&(deraruse!=mm))reinit=1;

 for(i=0;i<20;i++) tabgp[i]=0;

 while(val)
 {
```

```
   sauter=0;
   _llseek(ptr,seekpos,0);
   tabtetesl[it] = GlobalAlloc(GHND,(DWORD)sizeof tetesl);
   ptete = (t_tetesl *)GlobalLock( tabtetesl[it] );
   val = _lread(ptr, (LPSTR)ptete, sizeof tetesl);   cpt++;
   if ((val==0) || (ptete->etat == -1))
   {
    if(ptete->etat == -1) sauter=1;
    GlobalUnlock( tabtetesl[it] );
    GlobalFree( tabtetesl[it] );
    seekpos += sizeof tetesl + (tailuse * sizeof blocsl);
   }
   else
   {
    if (reinit) ptete->etat = 5;
    position[it] = cpt-1;
    if(ptete->etat==2)cpt_lt++;
    else tabgp[ ptete->ptgp ]++;
    GlobalUnlock( tabtetesl[it++] );
    seekpos += sizeof tetesl;
   }
   for(i=0;((val!=0)&&(i<tailuse)&&(sauter==0));i++)
   {
    _llseek(ptr,seekpos,0);
    tabblocsl[ib] = GlobalAlloc(GHND,(DWORD)sizeof blocsl);
    pbloc = (t_blocsl *)GlobalLock( tabblocsl[ib] );
    _lread(ptr,(LPSTR)pbloc, sizeof blocsl);
    GlobalUnlock( tabblocsl[ib++] );
    seekpos += sizeof blocsl;
   }
  }
}

void Tclassliste::init_scroll(int depart)
{
 int nb=0,j, locbloc, idze=0, indfile;  iec=0;

 for(int parcour=depart;((parcour<it)&&(nb<nbecran));parcour++)
 {
  ptete = (t_tetesl *)GlobalLock( tabtetesl[parcour] );
  if(ptete->etat != -1)
  {
   locbloc = parcour * tailuse;
   for(j=0;j<tailuse;j++)
   {
    pbloc = (t_blocsl *)GlobalLock( tabblocsl[locbloc] );
    ze[idze++]->SetText(pbloc->mot);
    GlobalUnlock( tabblocsl[locbloc] );
    locbloc++;
   }
   tabecran[iec++] = parcour;
   nb++;
  }
  GlobalUnlock( tabtetesl[parcour] );
 }
 if((parcour>=it)&&(nb<nbecran)) // place pour encoder nouvell s-l
 {
  if(it==0) indfile=0;
  else indfile=position[it-1]+1;
  while(nb<nbecran)
  {
   tabtetesl[it] = GlobalAlloc(GHND,(DWORD)sizeof tetesl);
   ptete = (t_tetesl *)GlobalLock( tabtetesl[it] );
   ptete->etat = 0;
   GlobalUnlock( tabtetesl[it] );
   tabecran[iec++] = it;
   position[it++] = indfile++;
   for(j=0;j<tailuse;j++)
    tabblocsl[ib++] = GlobalAlloc(GHND,(DWORD)sizeof blocsl);
   nb++;
  }
 }
 SetFocus(ze[0]->HWindow);
```

```
}

void Tclassliste::down() // it = premiere position vide dans la table tetesl
{
 comparer();

 haut = tabecran[iec-1]+1;
 init_scroll(haut);
}

void Tclassliste::up()
{
 comparer();

 int cpt=0; bas = tabecran[0]-1;
 while((cpt<nbecran)&&(bas>=0))
 {
  ptete = (t_tetesl *)GlobalLock( tabtetesl[ bas ] );
  if(ptete->etat != -1)
   cpt++;
  GlobalUnlock(tabtetesl[ bas ]);
  bas--;
 }
 bas++;

 init_scroll(bas);
}

void Tclassliste::comparer()
{
 int locbloc, idze=0, modif, j;    char recep[25];

 for(int i=0;i<iec;i++)
 {
  modif=0;
  locbloc = tabecran[i] * tailuse;
  for(j=0;j<tailuse;j++)
  {
   ze[idze]->GetText(recep,25);
   pbloc = (t_blocsl *)GlobalLock( tabblocsl[locbloc] );
   if (strcmp(pbloc->mot,recep)!=0) // il y a eu modif
   {
    strcpy(pbloc->mot,recep);
    modif=1;
   }
   GlobalUnlock( tabblocsl[locbloc] );
   ze[idze++]->SetText("");
   locbloc++;
  }
  if(modif)
  {
   ptete = (t_tetesl *)GlobalLock( tabtetesl[ tabecran[i] ] );
   if (ptete->etat < 5) ptete->etat += 5;
   GlobalUnlock(tabtetesl[ tabecran[i] ]);
  }
 }
}

void Tclassliste::sauver(int pf)
{
 long deplac;  int indbloc,j;

 if(pf==1)comparer();

 ptr=OpenFile(fichuse,&ofstr,OF_EXIST);
 if (ptr==-1) ptr=OpenFile(fichuse,&ofstr,OF_CREATE);
 else ptr=OpenFile(fichuse,&ofstr,OF_WRITE);

 for(int i=0;i<it;i++)
 {
  ptete = (t_tetesl *)GlobalLock( tabtetesl[i] );
  if(ptete->etat==-1)
  {
```

```
    deplac = position[i] * (sizeof tetesl + (tailuse * sizeof blocsl));
    _llseek(ptr,deplac,0);
    _lwrite(ptr,(LPSTR)ptete,sizeof tetesl);
   }
   if(ptete->etat >= 5)
   {
    ptete->etat -=5;
    deplac = position[i] * (sizeof tetesl + (tailuse * sizeof blocsl));
    _llseek(ptr,deplac,0);
    _lwrite(ptr,(LPSTR)ptete,sizeof tetesl);
    deplac += sizeof tetesl;
    indbloc = i * tailuse;
    for(int j=0;j<tailuse;j++)
    {
            pbloc = (t_blocsl *)GlobalLock( tabblocsl[indbloc] );
            _llseek(ptr,deplac,0);
            _lwrite(ptr,(LPSTR)pbloc,sizeof blocsl);
            deplac += sizeof blocsl;
            GlobalUnlock( tabblocsl[indbloc++] );
    }
   }
   indbloc = i*tailuse;
   for(j=0;j<tailuse;j++)
    GlobalFree( tabblocsl[indbloc++] );
   GlobalUnlock( tabtetesl[i] );
   GlobalFree( tabtetesl[i] );
  }
  _lclose(ptr);
}

void TEdition::trt_tab(RTMessage msg)
{
 switch(msg.WParam)
 {
  case VK_TAB : SetFocus( zsuiv );
                break;
  default : DefWndProc(msg);
 }
}

void TEdition::WMChar(RTMessage msg)
{
 switch(msg.WParam)
 {
  case  9 :
  case 13 : break;
  default : DefWndProc(msg);
 }
}

void Tclassliste::supprimer(void)
{
 int posintete = tabecran[selec];

 ptete = (t_tetesl *)GlobalLock( tabtetesl[posintete] );
 ptete->etat = -1;
 GlobalUnlock( tabtetesl[posintete] );

 init_scroll( tabecran[0] );
}

void Tclassliste::image(PTWindow pt)
{
 comparer();
 haut=tabecran[0];
 init_scroll(haut);

 int posintete = tabecran[selec];

 pt->GetApplication()->ExecDialog( new TD_Image(pt,this,posintete));
}

void Tclassliste::groupe(PTWindow pt)
```

```
{
 comparer();
 haut=tabecran[0];
 init_scroll(haut);

 int posintete = tabecran[selec];

 ptete = (t_tetesl *)GlobalLock( tabtetesl[posintete] );
 if(ptete->ptgp == 0)  //ptete a deja recu son pointeur qd on va ds TD
 {
  pt->GetApplication()->ExecDialog( new TD_groupe(pt,this) );
  if(ptete->etat<5) ptete->etat += 5;
 }
 else
  pt->GetApplication()->ExecDialog( new TD_refgrp(pt,this,ptete->ptgp) );
 GlobalUnlock( tabtetesl[posintete] );
}

void TD_Image::WMInitDialog(RTMessage)
{
 int posinbloc, id=401;

 posinbloc = (ptlis->tailuse) * posintete;
 for(int i=0;i<(ptlis->tailuse);i++)
 {
  ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
  hwnd = GetItemHandle(id++);
  SetWindowText(hwnd,ptlis->pbloc->mot);
  GlobalUnlock( ptlis->tabblocsl[posinbloc] );
  posinbloc++;
 }
 precedent=-1;
}

void TD_Image::push0(RTMessage)
{trt_image(390);}
void TD_Image::push1(RTMessage)
{trt_image(391);}
void TD_Image::push2(RTMessage)
{trt_image(392);}
void TD_Image::push3(RTMessage)
{trt_image(393);}
void TD_Image::push4(RTMessage)
{trt_image(394);}
void TD_Image::push5(RTMessage)
{trt_image(395);}
void TD_Image::push6(RTMessage)
{trt_image(396);}
void TD_Image::push7(RTMessage)
{trt_image(397);}
void TD_Image::push8(RTMessage)
{trt_image(398);}
void TD_Image::push9(RTMessage)
{trt_image(399);}
void TD_Image::push10(RTMessage)
{trt_image(400);}

void TD_Image::f_aide(RTMessage)
{
 GetApplication()->ExecDialog( new Thelp(5,pt) );
}

void TD_Image::trt_image(int cible)
{
 hwnd=GetItemHandle(411);
 GetWindowText(hwnd,texte,300);

 save_prec();

 if(cible==390)
 {
  ptlis->ptete = (Tclassliste::t_tetesl *)GlobalLock( ptlis->tabtetesl[posintete] );
  if((ptlis->ptete->ptimsl)!=0)
```

```
  {
   ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_READ);
   _llseek(ptrim,((ptlis->ptete->ptimsl)-1)*sizeof texte,0);
   _lread(ptrim,&texte,sizeof texte);
   SetWindowText(hwnd,texte);
   _lclose(ptrim);
  }
  else
   SetWindowText(hwnd,"");
  GlobalUnlock( ptlis->tabtetesl[ posintete ]);
 }
 else
 {
  posinbloc = (ptlis->tailuse) * posintete + cible - 391;
  ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
  if((ptlis->pbloc->ptim)!=0)
  {
   ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_READ);
   _llseek(ptrim,((ptlis->pbloc->ptim)-1)*sizeof texte,0);
   _lread(ptrim,&texte,sizeof texte);
   SetWindowText(hwnd,texte);
   _lclose(ptrim);
  }
  else
   SetWindowText(hwnd,"");
   GlobalUnlock( ptlis->tabblocsl[posinbloc] );
 }
 precedent=cible;

 SetFocus(hwnd);
}

void TD_Image::Ok(RTMessage)
{
 GetWindowText(hwnd,texte,300);
 save_prec();
 CloseWindow(IDOK);
}

void TD_Image::save_prec()
{
 if(precedent!=-1)
 {
  if(precedent==390)
  {
   ptlis->ptete = (Tclassliste::t_tetesl *)GlobalLock( ptlis->tabtetesl[posintete] );
   if((ptlis->ptete->ptimsl)!=0)
   {
          ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_WRITE);
    _llseek(ptrim,((ptlis->ptete->ptimsl)-1)*sizeof texte,0);
          _lwrite(ptrim,&texte,sizeof texte);
    _lclose(ptrim);
   }
   else
   {
          if(ptlis->ptete->etat<5) ptlis->ptete->etat +=5;
    ptlis->ptete->ptimsl = ecrire();
   }
   GlobalUnlock( ptlis->tabtetesl[ posintete ]);
  }
  else
  {
   posinbloc = (ptlis->tailuse) * posintete + precedent - 391;
   ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
   if((ptlis->pbloc->ptim)!=0)
   {
          ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_WRITE);
    _llseek(ptrim,((ptlis->pbloc->ptim)-1)*sizeof texte,0);
          _lwrite(ptrim,&texte,sizeof texte);
          _lclose(ptrim);
   }
   else
   {
```

```
            ptlis->ptete = (Tclassliste::t_tetesl *)GlobalLock( ptlis->tabtetesl[posintete] );
            if(ptlis->ptete->etat<5) ptlis->ptete->etat +=5;
            GlobalUnlock( ptlis->tabtetesl[ posintete ]);
            ptlis->pbloc->ptim = ecrire();
     }
     GlobalUnlock( ptlis->tabblocsl[posinbloc] );
    }
   }
}

int TD_Image::ecrire()
{
 long taille;
 ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_EXIST);
 if(ptrim==-1) ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_CREATE);
 else ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_READWRITE);
 _llseek(ptrim,0L,2);
 taille = tell(ptrim);
 taille /= sizeof texte;
 taille++;
 _lwrite(ptrim,&texte,sizeof texte);
 _lclose(ptrim);
 return(taille);
}

void TD_groupe::WMInitDialog(RTMessage)
{
 int val=0;

 hwnd=GetItemHandle(412);
 SendDlgItemMsg(412,CB_RESETCONTENT,0,0);
 SendDlgItemMsg(412,CB_LIMITTEXT,25,0);

 ptrgp=OpenFile(ptlis->gpuse,&ofstrgp,OF_EXIST);
 if(ptrgp!=-1)
 {
  ptrgp=OpenFile(ptlis->gpuse,&ofstrgp,OF_READ);
  _llseek(ptrgp,0L,0);
  val = _lread(ptrgp,(LPSTR)&titre, sizeof titre);
  while(val)
  {
   SendDlgItemMsg(412,CB_ADDSTRING,0,(DWORD)titre);
   val = _lread(ptrgp,(LPSTR)&titre, sizeof titre);
  }
  SendDlgItemMsg(412,CB_SETCURSEL,0,0);
  SetWindowText(hwnd,"");
  _lclose(ptrgp);
 }
}

void TD_groupe::Ok(RTMessage)
{
 DWORD pos, nb;

 ptrgp=OpenFile(ptlis->gpuse,&ofstrgp,OF_EXIST);
 if (ptrgp==-1)ptrgp=OpenFile(ptlis->gpuse,&ofstrgp,OF_CREATE);
 else ptrgp=OpenFile(ptlis->gpuse,&ofstrgp,OF_WRITE);

 GetWindowText(hwnd,titre,25);
 if (strcmp(titre,"")!=0)// le user a fait ok et a encode
 {
  nb = SendDlgItemMsg(412,CB_GETCOUNT,0,0);
  pos = SendDlgItemMsg(412,CB_GETCURSEL,0,0);
  if (pos==-1)// nouveau titre, put to end file
  {
   ptlis->ptete->ptgp = nb+1;
   _llseek(ptrgp,0L,2);
   _lwrite(ptrgp,&titre,sizeof titre);
  }
  else
   ptlis->ptete->ptgp = pos+1;
  CloseWindow(IDOK);
 }
```

```
 else
   SetFocus(hwnd);
 _lclose(ptrgp);
}

void TD_refgrp::WMInitDialog(RTMessage)
{
 hwnd = GetItemHandle(413);
 ptrgp=OpenFile(ptlis->gpuse,&ofstrgp,OF_READ);
 _llseek(ptrgp,(locgp-1)*sizeof titre,0);
 _lread(ptrgp,&titre,sizeof titre);
 SetWindowText(hwnd,titre);
 _lclose(ptrgp);
}

int Tclassliste::algo(PTWindow pt)
{
 int arret=0,iteration=1,nbpas,res;
 int nbit, indit, halt,init,cpt,i,iterat;

 init_algo();

 while((encore)&&(!arret))
 {
  cpt=0;  nbpas=0;
  for(i=0;i<20;i++)
  {
   tabpassage[i]=-1;
   if(iter[i]==0) cpt++;
  }
  if(cpt!=0)
  {
   for(i=0;(i<20)&&(nbpas<lg_empan);i++)
   {
          if(iter[i]==0) tabpassage[nbpas++]=i;
   }
   res = pt->GetApplication()->ExecDialog(new TD_affcouple(pt,this));
   if(res!=IDCANCEL)
   {
          if(mm==3)iterat=2;
          else iterat=1;
          while((iterat)&&(res!=IDCANCEL))
    {
           if(apprent==2)res=pt->GetApplication()->ExecDialog(new TD_reccouple(pt,this));
           if(apprent==1)res=pt->GetApplication()->ExecDialog(new TD_recpassif(pt,this));
           if(res==IDCANCEL) arret=1;
           else
           {
            if((mm==1)||(mm==2)) correction(iteration);
      else correctiongd(iteration);
           }
           iterat--;
    }
   }
   else
          arret=1;
  }
  for(i=0;i<20;i++) tabpassage[i]=-1;
  nbpas=0;
  for(i=0;i<20;i++)
  {
   if(iter[i]==iteration) tabpassage[nbpas++]=i;
  }
  if((res!=IDCANCEL)&&(nbpas!=0))
  {
   if(apprent==2)res=pt->GetApplication()->ExecDialog(new TD_reccouple(pt,this));
   if(apprent==1)res=pt->GetApplication()->ExecDialog(new TD_recpassif(pt,this));
   if(res==IDCANCEL) arret=1;
   else
   {
          if((mm==1)||(mm==2)) correction(iteration);
    else correctiongd(iteration);
   }
```

```
   }
   MessageBeep(0);
   iteration++;
  }
  return(arret);
}

void Tclassliste::init_algo()
{
  che=0;
  for(int i=0;i<20;i++)
  {
   affil[i]=0;
   affil2[i]=0;
   iter[i]=-1;
   tabalgo[i]=-1;
  }
  plibre=20;
  int j=0;
  for(i=0;(i<it)&&(j<20);i++)
  {
   ptete = (t_tetesl *)GlobalLock( tabtetesl[i] );
   if( ( (!gpex)&&(ptete->etat!=2) ) || ( (gpex)&&(ptete->etat!=2)&&(ptete->ptgp==indgp) ) )
   {
    if(j<20)
    {
            tabalgo[j]=i;
     iter[j++]=0;
            plibre--;
            if((gpex==1)&&(che<50))
            {
      if(ptete->ptgp == indgp) apgp[che++]=i;
            }
    }
   }
   GlobalUnlock( tabtetesl[i] );
   prochar = (i+1);
  }
  if (plibre==20)encore=0;
  else encore=1;
}

void Tclassliste::correction(int iteration)
{
  int valeur,inter;

  for(int i=0;tabpassage[i]!=-1;i++)
  {
   if(tabreponse[i]==0)
   {
    affil[ tabpassage[i] ]=0;
    iter[ tabpassage[i] ] = iteration+1;
   }
   else
   {
    if(tabreponse[i]==1) affil[ tabpassage[i] ]++;
    inter = affil[ tabpassage[i] ];
    valeur = incriter[frequence-1][inter];
    if(valeur!=-1)
            iter[ tabpassage[i] ] = iteration + valeur;
    else
    {
            iter[ tabpassage[i] ] = -1;
            envoi_lt(tabpassage[i]);
    }
   }
  }

  int j=0,k,cdt;
  if( (plibre>=lg_empan)&&(prochar<it) )
  {
   while(j<20)
   {
```

```
    if(iter[j]==-1)
    {
            k=j;
            while(k<19)
            {
             iter[k]=iter[k+1];
             affil[k]=affil[k+1];
       tabalgo[k]=tabalgo[k+1];
       k++;
            }
            iter[19]=-2;
            tabalgo[19]=-1;
    }
    else j++;
   }
   for(j=0;j<20;j++)
   {
    if(iter[j]==-2) iter[j]=-1;
   }
   for(int i=0;(i<20)&&(prochar<it);i++)
   {
    if(iter[i]==-1)
    {
     affil[i]=0;
     iter[i]=0;

            cdt=1;
            while((cdt==1)&&(prochar<it))
            {
             ptete = (t_tetesl *)GlobalLock( tabtetesl[prochar] );
             if( ( (!gpex)&&(ptete->etat!=2) ) || ( (gpex)&&(ptete->etat!=2)&&(ptete->ptgp==indgp) ) )
             {
              cdt=0;
              tabalgo[i]=prochar;
              plibre--;
              if((gpex==1)&&(che<50))
              {
         if(ptete->ptgp == indgp) apgp[che++]=i;
              }
       }
             GlobalUnlock( tabtetesl[prochar++] );
            }
    }
   }
  }
  if(plibre==20) encore=0;
}

void Tclassliste::correctiongd(int iteration)
{
 int valeur,inter;

 for(int i=0;tabpassage[i]!=-1;i++)
 {
  if(tabreponse[i]==0)
  {
   if(affil[ tabpassage[i] ] > affil2[ tabpassage[i] ])
            affil2[ tabpassage[i] ]=0;
   else
            affil[ tabpassage[i] ]=0;
   iter[ tabpassage[i] ] = iteration+1;
  }
  else
  {
   if(tabreponse[i]==1)
   {
            if(affil[ tabpassage[i] ] > affil2[ tabpassage[i] ])
             affil2[ tabpassage[i] ]++;
     else
             affil[ tabpassage[i] ]++;
   }
   if(affil[ tabpassage[i] ] > affil2[ tabpassage[i] ])
            inter=affil2[ tabpassage[i] ];
```

```
  else
        inter=affil[ tabpassage[i] ];
  valeur=incriter[frequence-1][inter];
  if(valeur!=-1)
   iter[ tabpassage[i] ] = iteration + valeur;
  else
  {
        if(affil2[ tabpassage[i] ]==frequence)
        {
         iter[ tabpassage[i] ] = -1;
         envoi_lt(tabpassage[i]);
        }
        else
        {
         iter[ tabpassage[i] ] = iteration + frequence;
        }
  }
 }
}
int j=0,k,cdt;
if( (plibre>=lg_empan)&&(prochar<it) )
{
 while(j<20)
 {
  if(iter[j]==-1)
  {
        k=j;
        while(k<19)
        {
         iter[k]=iter[k+1];
         affil[k]=affil[k+1];
         affil2[k]=affil2[k+1];
    tabalgo[k]=tabalgo[k+1];
    k++;
        }
        iter[19]=-2;
        tabalgo[19]=-1;
  }
  else j++;
 }
 for(j=0;j<20;j++)
 {
  if(iter[j]==-2) iter[j]=-1;
 }
 for(int i=0;(i<20)&&(prochar<it);i++)
 {
  if(iter[i]==-1)
  {
        affil[i]=0;
        affil2[i]=0;
        iter[i]=0;

        cdt=1;
        while((cdt==1)&&(prochar<it))
        {
         ptete = (t_tetesl *)GlobalLock( tabtetesl[prochar] );
         if( ( (!gpex)&&(ptete->etat!=2) ) || ( (gpex)&&(ptete->etat!=2)&&(ptete->ptgp==indgp) ) )
         {
          cdt=0;
          tabalgo[i]=prochar;
          plibre--;
          if((gpex==1)&&(che<50))
          {
       if(ptete->ptgp == indgp) apgp[che++]=i;
          }
    }
         GlobalUnlock( tabtetesl[prochar++] );
        }

  }
 }
}
if(plibre==20) encore=0;
```

```
}

void Tclassliste::envoi_lt(int ecran)
{
 ptete = (t_tetesl *)GlobalLock( tabtetesl[ tabalgo[ecran] ] );
 ptete->etat = 7;
 GlobalUnlock( tabtetesl[ tabalgo[ecran] ] );
 plibre++;
 cpt_ap++;
}

void TD_affcouple::WMInitDialog(RTMessage)
{
 int g=420, d=430, inter;
 for(int i=0;ptlis->tabpassage[i]!=-1;i++)
 {
  inter = ptlis->tabpassage[i];
  posinbloc = ptlis->tabalgo[ ptlis->tabpassage[i] ] * 2;
  if(ptlis->mm==2)posinbloc++;
  if(ptlis->mm==3)
  {if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc++;}
  ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
  hwnd = GetItemHandle(g++);
  SetWindowText(hwnd,ptlis->pbloc->mot);
  GlobalUnlock( ptlis->tabblocsl[posinbloc] );
  if(ptlis->mm==1) posinbloc++;
  if(ptlis->mm==2) posinbloc--;
  if(ptlis->mm==3)
  {
   if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc--;
   else  posinbloc++;
  }
  ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
  hwnd = GetItemHandle(d++);
  SetWindowText(hwnd,ptlis->pbloc->mot);
  GlobalUnlock( ptlis->tabblocsl[posinbloc] );
 }
}

void TD_reccouple::WMInitDialog(RTMessage)
{
 int borne=20,i,j,halt=0,inter;
 for(i=0;i<20;i++)
 {
  tabpos[i]=-1;
  ptlis->tabreponse[i] = ptlis->tabpassage[i];
  if((ptlis->tabpassage[i]==-1)&&(halt==0))
  {
   borne=i;
   halt=1;
  }
 }
 randomize();
 for(i=0;i<borne;i++)
 {
  halt=random(borne);
  j=halt;
  while(tabpos[j]!=-1)
  {
   j++;
   if(j==borne) j=0;
  }
  tabpos[j]=i;
 }
 for(i=0;i<borne;i++)
  ptlis->tabpassage[ tabpos[i] ] = ptlis->tabreponse[i];

 for(i=0;i<20;i++) ptlis->tabreponse[i]=0;

 hwnd=GetItemHandle(440);
 hwnd2=GetItemHandle(441);
 posinbloc = ptlis->tabalgo[ ptlis->tabpassage[0] ] * 2;
 inter=ptlis->tabpassage[0];
```

```
 if(ptlis->mm==2)posinbloc++;
 if(ptlis->mm==3)
 {if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc++;}
 ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
 SetWindowText(hwnd,ptlis->pbloc->mot);
 strcpy(donne,ptlis->pbloc->mot);
 GlobalUnlock( ptlis->tabblocsl[posinbloc] );
 ind=0; image=0;
 SetFocus(hwnd2);
}

void TD_reccouple::Ok(RTMessage)
{
 int inter = ptlis->tabpassage[ind];

 GetWindowText(hwnd2,reponse,25);
 if(ptlis->mm==1) posinbloc++;
 if(ptlis->mm==2) posinbloc--;
 if(ptlis->mm==3)
 {
  if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc--;
  else  posinbloc++;
 }
 ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
 if(strcmp(ptlis->pbloc->mot,reponse)==0)
 {
  if(image==0) ptlis->tabreponse[ind]=1;
  else ptlis->tabreponse[ind]=2;
 }
 else
 {
  ptlis->tabreponse[ind]=0;
  strcpy(autre,ptlis->pbloc->mot);
  pt->GetApplication()->ExecDialog( new TD_correct(pt,this) );
 }

 GlobalUnlock( ptlis->tabblocsl[posinbloc] );
 ind++;
 if((ind<20)&&(ptlis->tabpassage[ind]!=-1))
 {
  inter = ptlis->tabpassage[ind];
  posinbloc = ptlis->tabalgo[ ptlis->tabpassage[ind] ] * 2;
  if(ptlis->mm==2)posinbloc++;
  if(ptlis->mm==3)
  {if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc++;}
  ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
  SetWindowText(hwnd,ptlis->pbloc->mot);
  strcpy(donne,ptlis->pbloc->mot);
  GlobalUnlock( ptlis->tabblocsl[posinbloc] );
  SetWindowText(hwnd2,"");
  SetFocus(hwnd2);
 }
 else CloseWindow(IDOK);
 image=0;
}

void TD_reccouple::f_aide(RTMessage)
{
 GetApplication()->ExecDialog( new Thelp(4,pt) );
}

void TD_reccouple::imagecouple(RTMessage)
{
 if(ptlis->rev!=10)
 {
  image=1;
  pt->GetApplication()->ExecDialog(new TD_apimage(pt,2,posinbloc,ptlis,ind) );
 }
 else
  MessageBox(pt->HWindow,"Les associations mentales ne peuvent être utilisées pendant la révision","",MB_OK);
 SetFocus(hwnd2);

}
```

```
void TD_reccouple::imagemot(RTMessage)
{
 if(ptlis->rev!=10)
 {
  image=1;
  pt->GetApplication()->ExecDialog(new TD_apimage(pt,1,posinbloc,ptlis,ind) );
 }
 else
  MessageBox(pt->HWindow,"Les associations mentales ne peuvent être utilisées pendant la révision","",MB_OK);
 SetFocus(hwnd2);
}

void TD_correct::WMInitDialog(RTMessage)
{
 hwnd=GetItemHandle(444);
 SetWindowText(hwnd,ptrec->donne);
 hwnd=GetItemHandle(445);
 SetWindowText(hwnd,ptrec->autre);
}

void TD_recpassif::WMInitDialog(RTMessage)
{
 int halt=0, borne=20, inter;

 for(i=0;i<20;i++)
 {
  tabpos[i]=-1;
  ptlis->tabreponse[i] = ptlis->tabpassage[i];
  if((ptlis->tabpassage[i]==-1)&&(halt==0))
  {
   borne=i;
   halt=1;
  }
 }
 randomize();
 for(i=0;i<borne;i++)
 {
  halt=random(borne);
  j=halt;
  while(tabpos[j]!=-1)
  {
   j++;
   if(j==borne) j=0;
  }
  tabpos[j]=i;
 }
 for(i=0;i<borne;i++)
  ptlis->tabpassage[ tabpos[i] ] = ptlis->tabreponse[i];

 for(i=0;i<20;i++) ptlis->tabreponse[i]=0;

 SendDlgItemMsg(448,LB_SETCOLUMNWIDTH,250,0);
 for(i=0;(i<20)&&((ptlis->tabalgo[i])!=-1);i++)
 {
  posinbloc = ptlis->tabalgo[ i ] * 2;
  if(ptlis->mm==1)posinbloc++;
  ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[ posinbloc ] );
  SendDlgItemMsg(448,LB_ADDSTRING,0,(DWORD)ptlis->pbloc->mot);
  GlobalUnlock( ptlis->tabblocsl[ posinbloc ] );
 }
 if(ptlis->mm==3)
 {
  for(j=0;(j<20)&&((ptlis->tabalgo[j])!=-1);j++)
  {
   posinbloc = ptlis->tabalgo[ j ] * 2;
   posinbloc++;
   ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[ posinbloc ] );
   SendDlgItemMsg(448,LB_ADDSTRING,0,(DWORD)ptlis->pbloc->mot);
   GlobalUnlock( ptlis->tabblocsl[ posinbloc ] );
  }
 }
```

```
 hwnd=GetItemHandle(446);
 posinbloc = ptlis->tabalgo[ ptlis->tabpassage[0] ] * 2;
 inter = ptlis->tabpassage[0];
 if(ptlis->mm==2)posinbloc++;
 if(ptlis->mm==3)
 {if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc++;}
 ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
 SetWindowText(hwnd,ptlis->pbloc->mot);
 strcpy(donne,ptlis->pbloc->mot);
 GlobalUnlock( ptlis->tabblocsl[posinbloc] );
 ind=0; image=0;
 SendDlgItemMsg(448,LB_SETCURSEL,0,0);
 hwnd3=GetItemHandle(448);
 SetFocus(hwnd3);
}

void TD_recpassif::f_aide(RTMessage)
{
 GetApplication()->ExecDialog( new Thelp(3,pt) );
}

void TD_recpassif::trt_liste(RTMessage msg)
{
 switch(msg.LP.Hi)
 {
  case LBN_DBLCLK : clicok();
                    break;
 }
}

void TD_recpassif::Ok(RTMessage)
{
 clicok();
}

void TD_recpassif::imagecouple(RTMessage)
{
 if(ptlis->rev!=10)
 {
  image=1;
  pt->GetApplication()->ExecDialog(new TD_apimage(pt,2,posinbloc,ptlis,ind) );
 }
 else
  MessageBox(pt->HWindow,"Les associations mentales ne peuvent être utilisées pendant la révision","",MB_OK);
 SetFocus(hwnd3);
}

void TD_recpassif::imagemot(RTMessage)
{
 if(ptlis->rev!=10)
 {
  image=1;
  pt->GetApplication()->ExecDialog(new TD_apimage(pt,1,posinbloc,ptlis,ind) );
 }
 else
  MessageBox(pt->HWindow,"Les associations mentales ne peuvent être utilisées pendant la révision","",MB_OK);
 SetFocus(hwnd3);
}

void TD_corpas::WMInitDialog(RTMessage)
{
 hwnd=GetItemHandle(451);
 SetWindowText(hwnd,ptrecp->donne);
 hwnd=GetItemHandle(452);
 SetWindowText(hwnd,ptrecp->autre);
}

void TD_recpassif::clicok()
{
 int inter = ptlis->tabpassage[ind];

 sel = SendDlgItemMsg(448,LB_GETCURSEL,0,0);
 SendDlgItemMsg(448,LB_GETTEXT,sel,(DWORD)reponse);
```

```
if(ptlis->mm==1)posinbloc++;
if(ptlis->mm==2)posinbloc--;
if(ptlis->mm==3)
{
 if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc--;
 else  posinbloc++;
}
ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
if(strcmp(ptlis->pbloc->mot,reponse)==0)
{
 MessageBeep(0);
 if(image==0) ptlis->tabreponse[ind]=1;
 else ptlis->tabreponse[ind]=2;
}
else
{
 ptlis->tabreponse[ind]=0;
 strcpy(autre,ptlis->pbloc->mot);
 pt->GetApplication()->ExecDialog( new TD_corpas(pt,this) );
}
GlobalUnlock( ptlis->tabblocsl[posinbloc] );
ind++;
if((ind<20)&&(ptlis->tabpassage[ind]!=-1))
{
 inter = ptlis->tabpassage[ind];
 posinbloc = ptlis->tabalgo[ ptlis->tabpassage[ind] ] * 2;
 if(ptlis->mm==2)posinbloc++;
 if(ptlis->mm==3)
 {if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc++;}
 ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
 SetWindowText(hwnd,ptlis->pbloc->mot);
 strcpy(donne,ptlis->pbloc->mot);
 GlobalUnlock( ptlis->tabblocsl[posinbloc] );
 SetFocus(hwnd3);
}
else CloseWindow(IDOK);
image=0;
}

void TD_apimage::WMInitDialog(RTMessage)
{
 hwndim=GetItemHandle(453);

 int inter = ptlis->tabpassage[ind];

 if(tim==1) //imagemot
 {
  if(ptlis->mm==1)posinbloc++;
  if(ptlis->mm==2)posinbloc--;
  if(ptlis->mm==3)
  {
   if(ptlis->affil[inter] > ptlis->affil2[inter]) posinbloc--;
   else  posinbloc++;
  }
  ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
  if((ptlis->pbloc->ptim)!=0) // existe im
  {
   ptrim = OpenFile(ptlis->imuse,&ofstrim,OF_READ);
   _llseek(ptrim,((ptlis->pbloc->ptim)-1)*sizeof texte,0);
   _lread(ptrim,&texte,sizeof texte);
   SetWindowText(hwndim,texte);
   _lclose(ptrim);
  }
  GlobalUnlock( ptlis->tabblocsl[posinbloc] );
 }
 else
 {
  if(ptlis->mm==2)posinbloc--;
  posintete=posinbloc/2;
  ptlis->ptete = (Tclassliste::t_tetesl *)GlobalLock( ptlis->tabtetesl[posintete] );
  if((ptlis->ptete->ptimsl)!=0)//existe im
  {
   ptrim = OpenFile(ptlis->imuse,&ofstrim,OF_READ);
```

```
    _llseek(ptrim,((ptlis->ptete->ptimsl)-1)*sizeof texte,0);
    _lread(ptrim,&texte,sizeof texte);
    SetWindowText(hwndim,texte);
    _lclose(ptrim);
   }
   GlobalUnlock( ptlis->tabtetesl[posintete] );
  }
  SetFocus(hwndim);
}

void TD_apimage::Ok(RTMessage)
{
 //posinbloc pointe sur le 2ime mot du couple ou le premier
 GetWindowText(hwndim,texte,300);
 if(tim==1)//imagemot
 {
  ptlis->pbloc = (Tclassliste::t_blocsl *)GlobalLock( ptlis->tabblocsl[posinbloc] );
  if((ptlis->pbloc->ptim)!=0) // existe im
  {
   ptrim = OpenFile(ptlis->imuse,&ofstrim,OF_WRITE);
   _llseek(ptrim,((ptlis->pbloc->ptim)-1)*sizeof texte,0);
   _lwrite(ptrim,&texte,sizeof texte);
   _lclose(ptrim);
  }
  else//image inexistante
  {
   if(strcmp(texte,"")!=0)
   {
    ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_EXIST);
    if(ptrim==-1) ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_CREATE);
          else ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_READWRITE);

          _llseek(ptrim,0L,2);
    taille = tell(ptrim);
          taille = taille / sizeof texte;

          _lwrite(ptrim,&texte,sizeof texte);
          _lclose(ptrim);

    taille++;
          ptlis->pbloc->ptim = taille;
          posintete=posinbloc;
          if(ptlis->mm==1)posintete--;
          posintete/=2;
    ptlis->ptete = (Tclassliste::t_tetesl *)GlobalLock( ptlis->tabtetesl[posintete] );
    if(ptlis->ptete->etat < 5) ptlis->ptete->etat += 5;
    GlobalUnlock( ptlis->tabtetesl[posintete] );
   }
  }
  GlobalUnlock( ptlis->tabblocsl[posinbloc] );
 }
 else//image couple
 {
  ptlis->ptete = (Tclassliste::t_tetesl *)GlobalLock( ptlis->tabtetesl[posintete] );
  if((ptlis->ptete->ptimsl)!=0)//existe im
  {
   ptrim = OpenFile(ptlis->imuse,&ofstrim,OF_WRITE);
   _llseek(ptrim,((ptlis->ptete->ptimsl)-1)*sizeof texte,0);
   _lwrite(ptrim,&texte,sizeof texte);
   _lclose(ptrim);
  }
  else//image inexistante
  {
   if(strcmp(texte,"")!=0)
   {
    ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_EXIST);
    if(ptrim==-1) ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_CREATE);
          else ptrim=OpenFile(ptlis->imuse,&ofstrim,OF_READWRITE);

          _llseek(ptrim,0L,2);
    taille = tell(ptrim);
          taille = taille / sizeof texte;
```

```
            _lwrite(ptrim,&texte,sizeof texte);
            _lclose(ptrim);

      taille++;
            ptlis->ptete->ptimsl = taille;
      if(ptlis->ptete->etat < 5) ptlis->ptete->etat += 5;
     }
    }
    GlobalUnlock( ptlis->tabtetesl[posintete] );
  }
  CloseWindow(IDOK);
}

int Tclassliste::revision(PTWindow pt)
{
  int i=0,j,k,res=1,virg=1;
  rev=10;
  while((i<it)&&(res!=IDCANCEL))
  {
   for(j=0;j<20;j++)
   {
    tabalgo[j]=-1;
    tabpassage[j]=-1;
    affil[j]=0;
    affil2[j]=0;
   }
   j=0;
   while( (i<it)&&(j<20) )
   {
    ptete = (t_tetesl *)GlobalLock( tabtetesl[ i ] );
    if(ptete->etat == 2)
    {
     tabalgo[j]=i;
            tabpassage[j]=j;
            virg=0; j++;
    }
    GlobalUnlock( tabtetesl[i++] );
   }
   if(j>0)
   {
    if(mm!=3)
    {
            if(apr==2)res=pt->GetApplication()->ExecDialog(new TD_reccouple(pt,this));
     if(apr==1)res=pt->GetApplication()->ExecDialog(new TD_recpassif(pt,this));
            if(res!=IDCANCEL)correv();
    }
    else
    {
     if(apr==2)res=pt->GetApplication()->ExecDialog(new TD_reccouple(pt,this));
     if(apr==1)res=pt->GetApplication()->ExecDialog(new TD_recpassif(pt,this));
     if(res!=IDCANCEL)
     {
      correv();
      for(k=0;k<20;k++) affil[k]++;
      if(apr==2)res=pt->GetApplication()->ExecDialog(new TD_reccouple(pt,this));
      if(apr==1)res=pt->GetApplication()->ExecDialog(new TD_recpassif(pt,this));
           if(res!=IDCANCEL)correv();
     }
    }
   }
   else
   {
    if(virg==1)MessageBox(pt->HWindow,"Les mots de cette liste n'ont pas encore étés étudiés.","",MB_OK);
   }
  }
  rev=0;
  return(virg);
}

void Tclassliste::correv()
{
  for(int i=0;(i<20)&&(tabalgo[i]!=-1);i++)
  {
```

```
   if(tabreponse[i]==0)
   {
    ptete = (t_tetesl *)GlobalLock( tabtetesl[ tabalgo[ tabpassage[i] ] ] );
    ptete->etat = 5;
    GlobalUnlock( tabtetesl[ tabalgo[ tabpassage[i] ] ] );
   }
   else
    cpt_ap++;
  }
}

void TD_result::WMInitDialog(RTMessage)
{
 inter = ptlis->cpt_lt + ptlis->cpt_ap;
 pourcent = (100 * inter) / (ptlis->it);
 hwnd = GetItemHandle(458);
 sprintf(tamp,"%d",pourcent);
 SetWindowText(hwnd,tamp);

 hwnd = GetItemHandle(459);
 sprintf(tamp,"%d",ptlis->cpt_ap);
 SetWindowText(hwnd,tamp);

 hwnd = GetItemHandle(460);
 inter = ptlis->it - ptlis->cpt_lt - ptlis->cpt_ap;
 sprintf(tamp,"%d",inter);
 SetWindowText(hwnd,tamp);
}

void TD_resurev::WMInitDialog(RTMessage)
{
 hwnd = GetItemHandle(461);
 sprintf(tamp,"%d",ptlis->cpt_lt);
 SetWindowText(hwnd,tamp);

 if(ptlis->mm == 3) ptlis->cpt_ap /= 2;

 hwnd = GetItemHandle(462);
 sprintf(tamp,"%d",ptlis->cpt_ap);
 SetWindowText(hwnd,tamp);
}

void TD_lgempan::WMInitDialog(RTMessage)
{
 SendDlgItemMsg(463,CB_RESETCONTENT,0,0);
 for(int i=2;i<8;i++)
 {
  itoa( i,tamp,10);
  SendDlgItemMsg(463,LB_ADDSTRING,0,(DWORD)tamp);
 }
 SendDlgItemMsg(463,LB_SETCURSEL,3,0);
}

void TD_lgempan::Ok(RTMessage)
{
 ptlis->lg_empan = SendDlgItemMsg(463,LB_GETCURSEL,0,0);
 ptlis->lg_empan += 2;
 CloseWindow(IDOK);
}

void Tclassliste::rec_gp(PTWindow pt)
{
 pt->GetApplication()->ExecDialog( new TD_recapit(pt,this) );
}

void TD_recapit::WMInitDialog(RTMessage)
{
 hwnd = GetItemHandle(466);
 SetWindowText(hwnd,ptlis->titre);

 SendDlgItemMsg(465,LB_SETCOLUMNWIDTH,250,0);
 for(int i=0;(i< ptlis->it);i++)
 {
```