Les structures itératives

Vous allez pouvoir constater que la traduction d'un algorithme itératif à l'aide d'un langage de programmation événementiel n'est pas toujours évident. Il est parfois nécessaire d'éclater le code pour obtenir un comportement satisfaisant (ça en fera au moins un qui s'éclate…). En fait, cet « éclatement » est nécessaire lorsqu'il y a une (ou des) entrée/sortie dans l'itération (c'est-à-dire une saisie utilisateur, une lecture dans un fichier, un affichage, une écriture dans un fichier… bref, quelque chose qui vient de l'extérieur, ou qui y va).

Le nombre d'exercices de cette séquence est très important. On ne va pas créer 29 fenêtres… enfin… ceux qui le souhaitent peuvent le faire, bien sûr, mais pour ma part, je ne vous fournis pas l'aide pour programmer tous les algos : j'ai choisi les plus significatifs dans chaque partie.

Je veux profiter de cette séquence pour, qu'au lieu de faire un menu déroulant, vous présentiez chacun de vos programmes sur un onglet. Les onglets… mais si, vous savez bien… ces trucs qu'on met en avant-plan sur un simple clic. Si vous avez oublié comment on fait avec Windev, reprenez le manuel d'autoformation.

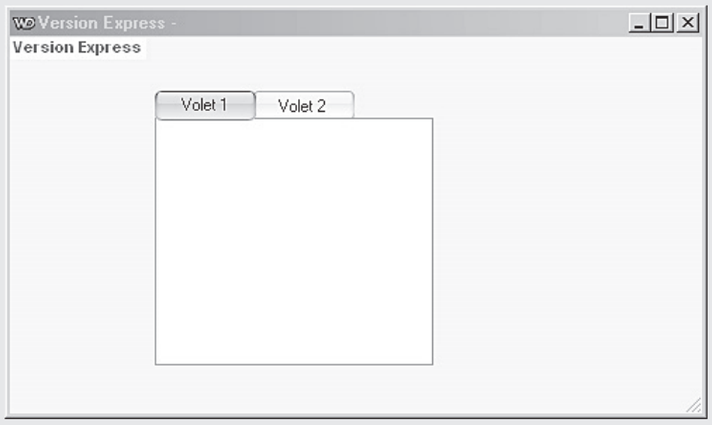
Je voudrais aussi que sur quelques onglets on écrive une aide, une vraie aide Windows, comme celle que nous-même utilisons quand on consulte l'aide d'un logiciel Windows.

Au boulot.

Consigne 1 : créez un nouveau projet pour cette 4e séquence.

Notre projet ne contiendra qu'une seule fenêtre puisqu'on mettra chaque exo sur un plan d'onglet.

Créez cette fenêtre et posez dessus un objet onglet (icône dont la bulle d'aide est **créer un onglet** dans la barre d'outils).



Ci-dessous, je reprends le cours et les exercices, de la même façon qu'au cours du TP précédent.

Partie 1

La "boucle" Pour…FinPour

1. Présentation de la structure

Allez faire un tour dans l'aide de Windev et tapez **pour** dans l'index de l'aide.

Je reproduis ici les extraits importants de cette page d'aide sur la structure « Pour » :

Syntaxe d'une instruction Pour	
En algo	**En Windev**
Pour var de valeurinit à valeurfinale (augmenter ou diminuer de x) Faire // instructions FinPour	**POUR** var = valeurinit à valeurfinale [**PAS** x] // instructions **FIN**

Il n'y a pas de syntaxe particulière pour une itération décroissante, simplement dans ce cas, **x** doit être négatif.

Vous avez remarqué que dans Windev **pas x** (c'est le pas d'incrémentation : la quantité qu'on ajoute ou qu'on retranche à chaque tour de boucle) est entre crochets, ce qui signifie que cette précision est optionnelle :

- si on ne met rien, cela équivaut en algo à **(augmenter de 1) ;**
- si on met **PAS x** et que x est positif, cela équivaut à **(augmenter de x) ;**
- si on met **PAS x** et que x est négatif, cela équivaut à **diminuer de la valeur absolue de x**

À part ça, la structure **POUR** de Windev se comporte comme il a été décrit dans le cours d'algo.

Exemple 1

```
POUR i = 1 à 10
info (i)
FIN
```

Exemple 2

```
POUR i = 10 à 1 pas -1
info (i)
FIN
```

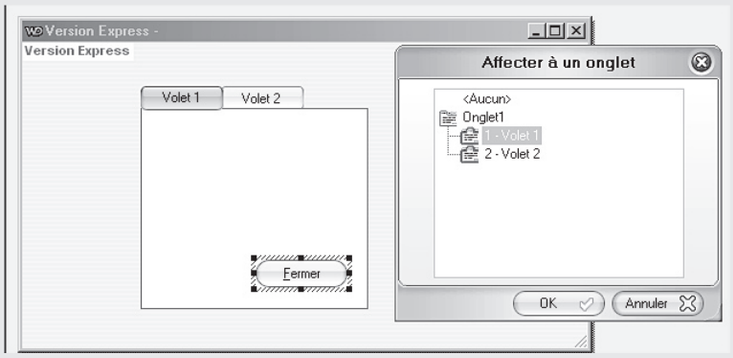
Revenons maintenant à la programmation des exercices.

Dans ce projet, chaque plan de l'onglet devra comporter le même bouton Fermer permettant de fermer l'application.

Pour obtenir ce résultat, c'est tout simple. Posez un bouton **boutonFermer**, de libellé **Fermer**.

Ensuite, ce bouton étant sélectionné, faites un clic droit. Dans le menu contextuel qui s'affiche, choisissez **Associer à un onglet**.

Une petite fenêtre s'ouvre, de titre **Affecter à un onglet**. Dans cette fenêtre, choisissez **<Aucun>**, puis validez.



Testez votre fenêtre. Vous pouvez constater que votre bouton **Fermer** est visible sur le **Volet 1** et le **Volet 2** de votre onglet.

En ce qui concerne ce bouton **Fermer**, il ne vous reste plus qu'à mettre, dans son événement clic, le code provoquant la fermeture de l'application (il s'agit bien sûr de l'instruction **Ferme**).

Allez, on attaque les exercices.

Exercice 31

Écrire l'algo de calcul et d'affichage de la moyenne de x élèves à un devoir, x étant saisi.

Celui là, je vais le faire avec vous. Il va nous permettre de nous rendre rapidement compte que la traduction d'une itération n'est pas toujours évidente avec un langage événementiel.

Ce qu'il faut, c'est créer un traitement itératif permettant de saisir autant de notes qu'il y a d'élèves, et faire le cumul des notes à chaque fois qu'une note a été saisie.

Dans le TP précédent (TP de la séquence 3), nous avons été confrontés aux problèmes de traduction de l'algo de l'exercice 26 (jeu du mot à trouver) à l'aide d'un langage événementiel. Ces difficultés étaient issues du fait que cet algo était en quelque sorte itératif puisque le joueur avait la possibilité de saisir 3 essais successifs.

À chaque fois qu'on aura à traduire un algo itératif avec des saisies dans l'itération, on rencontrera cette même situation.

Replongez-vous quelques minutes (plouf !) dans les explications de l'exo 26 du TP de la séquence 3. Je vous y ai proposé 2 solutions de développement, la première restant très proche de l'algo, et la seconde, plus ergonomique, mais nécessitant d'éclater le code dans différents événements de différents objets de la fenêtre.

Nous sommes ici dans ce cas.

Réfléchissons zensemble au meilleur comportement à donner à ce traitement de calcul de moyenne.

> Soit, pour la saisie de chaque note, on affiche une petite boîte de dialogue où l'utilisateur(trice) saisit la note courante, que l'on récupère et que l'on cumule. On affiche cette boîte de dialogue autant de fois qu'il y a de notes. Ensuite, on calcule la moyenne que l'on affiche dans une zone de texte sur notre onglet ou dans un message. C'est bof bof, comme comportement : il n'y aura pratiquement rien sur notre onglet **Moyenne**, et l'utilisateur va voir je ne sais combien de boîtes de dialogues s'afficher successivement.
>
> Ou alors, on permet à l'utilisateur de saisir la note courante dans un champ de saisie sur l'onglet, dès qu'il a saisi une note, on affiche la moyenne provisoire dans une zone de texte prévue à cet effet sur notre onglet. L'inconvénient de ce fonctionnement est que l'utilisateur aura du mal à se repérer concernant le nombre de notes déjà saisies.
>
> Je trouve qu'il serait plus opportun de n'afficher la moyenne que quand il aura saisi le nombre prévu de notes, soit dans un message, soit mieux, dans une zone de texte que l'on peut ne rendre visible sur l'onglet qu'au moment de l'affichage de la moyenne définitive.
>
> On peut aussi lui rajouter une information lui permettant à chaque instant de savoir où il en est dans sa saisie, une information du style **Vous avez saisi x notes sur y**.

Oui voilà, on va faire comme ça :

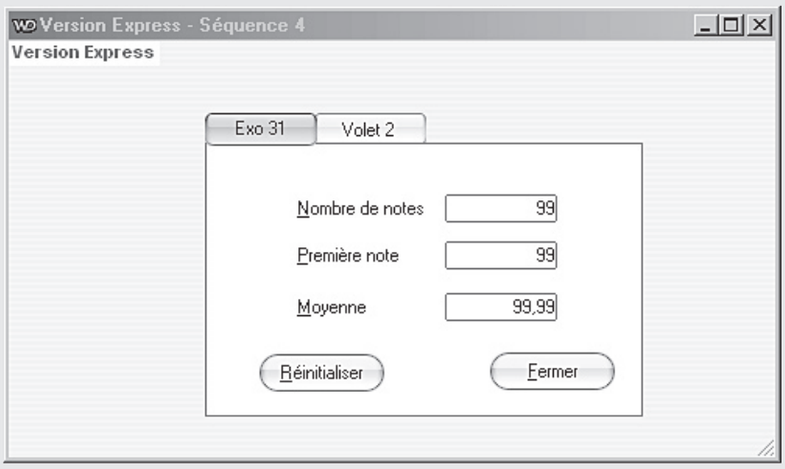
- on permet à l'utilisateur de saisir la note courante dans un champ de saisie sur l'onglet ;

- on n'affiche la moyenne que quand il aura saisi le nombre prévu de notes, dans une zone de texte que l'on ne rend visible sur l'onglet qu'au moment de l'affichage de la moyenne définitive ;
- on lui rajoute une information lui permettant à chaque instant de savoir où il en est dans sa saisie, une information du style **Vous avez saisi x notes sur y** ;
- et enfin, cerise sur le gâteau, on prévoit un bouton permettant de réinitialiser la fenêtre pour calculer une nouvelle moyenne.

Alors maintenant qu'on est au clair avec ce qu'on veut, on va commencer par faire la maquette de notre onglet : on pose les objets sur l'onglet (on crée le « look »).

Ensuite, on écrira les traitements : on créera donc le comportement (le « feel »). Cette expression « look and feel » est très prisée chez les puristes de l'ergonomie. Vous voilà vous aussi au parfum. Bon OK, pas facile à replacer dans une conversation entre copains.

Le plan de l'onglet correspondant à cet exercice doit avoir à peu près le look suivant :



Ci-dessus, le volet **Exo 31** a l'apparence qu'il doit avoir lorsque l'utilisateur l'active ou le réinitialise, c'est-à-dire qu'il a l'apparence qu'il doit avoir lorsque l'utilisateur s'apprête à effectuer la saisie d'une nouvelle série de notes.

Pour donner à ce volet le libellé **Exo31**, faites un clique droit sur **Volet 1** puis, dans le menu contextuel qui s'affiche, choisissez **Editer le libellé**. Il ne vous reste plus ensuite qu'à taper le libellé de votre choix.

Les champs de saisie **Nombre de notes**, **Première note** et **Moyenne** s'appellent respectivement **saisieNbNotes**, **saisieNote** et **saisieMoyenne**.

Le champ de saisie **Première note** changera de libellé au cours de l'exécution du traitement : la première note une fois saisie, ce libellé deviendra **Note n°2**, puis ensuite **Note n°3** et ainsi de suite…

Maintenant qu'on s'est occupé du look, occupons-nous du feel.

Le champ de saisie **Moyenne** ne doit apparaître qu'au moment de l'affichage de la moyenne, donc, dans l'onglet **IHM**, dans la zone **Etat initial**, je décoche **Visible**. Nous rendrons le champ de saisie **Moyenne** visible lorsque toutes les notes auront été saisies.

Comme je l'ai rapidement expliqué quelques lignes plus haut, le libellé **Première note** sera à modifier au fur et à mesure des saisies utilisateur : on lui affichera quelque chose comme **2e note**, puis **3e note** et ainsi de suite.

Le calcul de la moyenne est à mettre dans l'événement de sortie de la zone où l'utilisateur saisit les notes.

Le problème est qu'on ne peut pas déclarer et initialiser le cumul des notes et le compteur d'itérations dans cet événement **Sortie de saisieMoyenne** car si on déclare ces 2 variables dans cet événement, elles seront réinitialisées à chaque fois que l'utilisateur valide la saisie d'une note. Or, le cumul et le compteur d'itérations voient leurs valeurs évoluer au cours de ce calcul de moyenne.

Donc, il faut les déclarer et les initialiser dans l'événement d'un objet hiérarchiquement « au dessus » du champ de saisie **saisieMoyenne** : on pourrait les déclarer et les initialiser soit dans un événement de l'onglet, soit dans un événement de la fenêtre.

Posons-nous la question suivante : quand le cumul des notes et le compteur d'itérations ont-ils besoin d'être initialisés ou réinitialisés ?

- À l'ouverture de la fenêtre (donc au lancement de l'application).
- Lorsque l'utilisateur clique sur le bouton **Réinitialiser** du volet **Exo 31.**
- Lorsque l'utilisateur clique sur le volet **Exo 31**, alors qu'il utilisait un autre volet.

Est-il opportun de déclarer et initialiser le cumul des notes et le compteur d'itérations dans un événement associé à l'onglet ? lorsqu'on va dans le code de l'onglet, on constate que les deux seuls événements attaché à un onglet sont les événements **Initialisation** et **A chaque modification**, et ces événements concernent l'onglet, pas un volet particulier de l'onglet.

Je vais dans l'index de l'aide de windev, je tape **Onglet**, puis je choisis **Traitements associés aux onglets**.

En lisant l'aide, j'apprends que l'événement **Initialisation** se produit lors de l'ouverture de la fenêtre supportant l'onglet, c'est-à-dire que cet événement se produit, pour nous, au lancement de l'application, puisque celle-ci ne comporte qu'une seule fenêtre.

Nous devons donc y déclarer et initialiser nos deux variables,pour la première utilisation du programme de calcul de moyenne, après lancement de l'application.

Si, par contre, l'utilisateur utilise une deuxième fois le programme de calcul de moyenne sans avoir refermé puis rouvert la fenêtre (donc l'application), le cumul des notes et le compteur d'itérations ne seront pas réinitialisés.

Je lis maintenant les renseignements concernant l'événement **A chaque modification**. Cet événement se produit à chaque fois que l'utilisateur choisit un autre volet d'onglet. Notre volet **Exo31** étant le premier, l'utilisateur ne provoquera pas l'événement à chaque modification de façon systématique. L'utilisateur provoquera l'événement **A chaque modification** lorsqu'il cliquera pour remettre en avant plan ce volet d'onglet, et à ce moment là, nos variables devront être réinitialisées.

Nous devons donc réinitialiser nos deux variables dans le code associé à cet événement.

Mais ce n'est pas encore suffisant. En effet, le cumul des notes et le compteur d'itérations doivent également être réinitialisés lors du clic sur le bouton **Réinitialiser**.

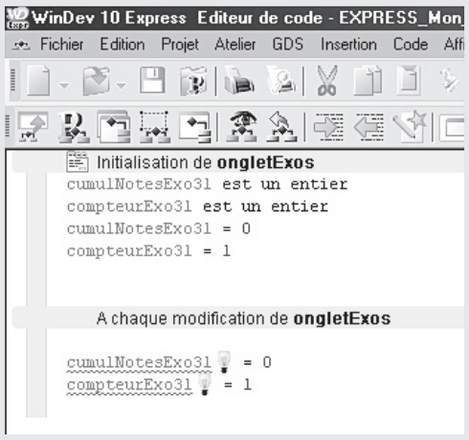
Donc, récapitulons :

- nous allons déclarer et initialiser le cumul des notes et le compteur d'itérations dans l'événement **Initialisation** de notre onglet ;
- nous réinitialiserons ces deux variables dans les événement **A chaque modification** de l'onglet et **clic** sur le bouton **Réinitialiser**.

Comme dans l'événement **Initialisation** de l'onglet, nous allons probablement déclarer d'autres variables pour les autres volets, j'indique, pour chaque variable, quel volet du plan d'onglet elle concerne.

J'appelle donc le cumul des notes **cumulNotesExo31** et le compteur d'itérations **compteurExo31**.

Voici le code relatif à la déclaration et à l'initialisation de nos deux variables :



Vous constatez vous-même ci-dessus qu'il y a un problème. En effet, lorsque je déclare mes deux variables dans l'événement **Initialisation** de notre onglet, l'événement **A chaque modification** de l'onglet ne reconnaît pas ces variables, qui, en fait, sont locales à l'événement **Initialisation**.

Donc, je dois changer mes déclarations de place, afin que mes deux variables soient visibles depuis les deux événements, et aussi depuis le **clic** sur le bouton **Réinitialiser**.

Donc, je coupe et je colle mes deux déclarations **cumulNotesExo31 est un entier** et **compteurExo31 est un entier** dans la partie **Déclarations globales** de la fenêtre, je recompile le projet en allant dans **Projet** l **Compiler le projet** (car j'ai constaté que parfois, windev ne le fait pas tout seul et conserve des erreurs antérieures si je ne recompile pas).

Cette fois, mes instructions **cumulNotesExo31 = 0** et **compteurExo31 = 1**, qui se trouvent dans les événements : **Initialisation de ongletExos**, **A chaque modification de ongletExos** et **Clic sur boutonReinitialiser** ne provoquent plus de message d'erreur.

Bon, on avance.

Je souhaite maintenant qu'à l'ouverture de la fenêtre ou bien lors de la sélection par un clic du volet **Exo31**, le champ de saisie **Nombre de note** ait le focus. Donc, dans l'événement **Initialisation** de mon onglet, je rajoute l'instruction **RepriseSaisie(saisieNbNotes)**.

J'en profite pour régler correctement l'ordre d'accessibilité de mes champs (je ne vous aide pas, vous savez faire).

Ensuite, il reste à programmer l'événement de sortie du champ de saisie des notes, que j'ai appelé **saisieNote**.

Nous devons y simuler une itération mais de l'itération, nous ne devons garder que le fait que l'on cumule les notes si le nombre de note déjà saisies n'a pas encore dépassé le nombre effectif de notes… Vous me suivez là ?

En fait, de l'itération, on ne garde que le test : si le compteur n'est pas encore égal au nombre effectif de notes, alors on permet encore la saisie d'une note, sinon, c'est que le bon nombre de notes a été saisi, et alors on calcule et on affiche la moyenne.

Voici le code tel que je l'imagine. Nous testerons ensuite ce qui fonctionne ou non. Tapez vous aussi ce code dans l'événement **Sortie de saisieNote.**

```
// Si il reste des notes à saisir
SI compteurExo31 <> saisieNbNotes
// On rajoute au compteur la note qui vient d'être saisie.
        compteurExo31 = compteurExo31 + 1
        // On rajoute cette note au cumul
        cumulNotesExo31 = cumulNotesExo31 + saisieNote
        // On vide le champ de saisie et on redonne le focus au champ
        saisieNote = ""
        RepriseSaisie ("saisieNote")
        // On adapte le libellé au rang de la note à saisir
        saisieNote..Libellé = compteurExo31 + "° note "
SINON
        // On calcule et on affiche la moyenne
        saisieMoyenne = cumulNotesExo31 / saisieNbNotes
        // On rend le champ saisieMoyenne visible
        saisieMoyenne..Visible = Vrai
FIN
```

On teste.

Pour faire mon test, je saisis **3** dans **Nombre de notes**, puis je saisis, comme notes, **12**, **11** et **10**, je valide.

Premier problème : la moyenne qui s'affiche est de **7.67**, elle est fausse.

En réfléchissant, on se rend vite compte que cette moyenne divise par 3 la somme des deux premières notes saisies, ce qui est normal, puisque dans le code, on ne rajoute pas la note saisie au cumul lorsque **compteurExo31 = saisieNbNotes**.

En fait, l'instruction **cumulNotesExo31 = cumulNotesExo31 + saisieNote** est mal placée. Elle n'a rien à faire dans le **SI**, car la note saisie doit dans tous les cas être rajoutée au cumul.

Je change cette instruction de place, je la mets avant le **SI**.

Je teste à nouveau. La moyenne calculée est exacte.

Second problème : le libellé n'est pas actualisé, il reste "coincé" à la valeur **Première note**.

Par curiosité, je modifie l'instruction **saisieNote..Libellé = compteurExo31 + "° note "** qui devient **saisieNote..Libellé = "° note ".** Et cette fois, lorsqu'on teste, le libellé change. Le problème vient donc de la partie **compteurExo31** de l'instruction.

J'essaie **saisieNote..Libellé = NumériqueVersChaîne(compteurExo31)+ "° note "**, et cette fois ça fonctionne. Finalement, c'était normal que ça ne fonctionne pas, car on essayait de concaténer un nombre entier avec une chaîne de caractères!!

Il nous reste ensuite à programmer correctement le **clic** sur le bouton **Réinitialiser**. Pour l'instant, dans cet événement, on se contente de réinitialiser le compteur de notes et le cumul. Que doit-on y mettre ? Il faut remettre à vide toutes les zones de saisie, rendre **Moyenne** à nouveau invisible, remettre à 0 le compteur de notes et le cumul et redonner le focus au champ Nombre de notes.

Je vous conseille de compléter le traitement d'initialisation de l'onglet comme suit…

```
cumulNotesExo31 = 0
compteurExo31 = 1
saisieNbNotes = ""
saisieNote = ""
saisieMoyenne = ""
saisieMoyenne..Visible = Faux
RepriseSaisie(saisieNbNotes)
```

… et d'appeler ce traitement d'initialisation dans le **clic sur boutonReinitialiser** et dans **A chaque modification de ongletExos**, desquels vous aurez retiré les initialisations qu'on y avait mises pour les remplacer par l'instruction suivante, qui appelle le traitement d'initialisation.

```
ExécuteTraitement(ongletExos,trtInit)
```

Voilà !!

L'instruction **RepriseSaisie(saisieNbNotes)** doit bien être en dernier car elle a pour effet d'interrompre le traitement en cours et d'exécuter le traitement d'entrée du champ spécifié, ici, le champ **saisieNbNotes**.

Exercice 32

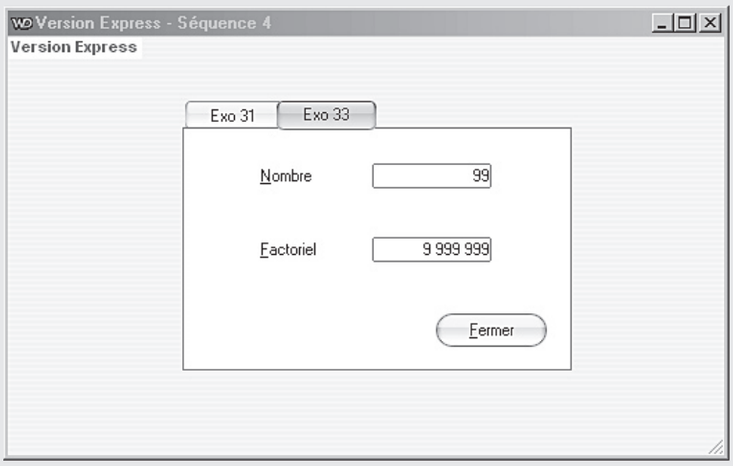
On le passe, vous pouvez modifier l'exo 31 pour qu'il devienne l'exo 32.

Exercice 33

Écrire l'algo qui calcule et affiche le factoriel d'un nombre saisi au clavier.

Celui-là, on va le faire pour que vous vous rendiez compte que quand il n'y a pas de saisie dans l'itérative, la traduction des traitements itératifs est très proche de l'algo.

Voici l'interface graphique de ce volet d'onglet :



Voici le code, tapé dans l'événement de sortie du champ **saisieNb** (dans lequel l'utilisateur saisit le nombre dont il veut le factoriel). Le champ où on affiche le factoriel s'appelle **saisieFact** :

```
compteur est un entier
saisieFact = saisieNb
POUR compteur = saisieNb - 1 à 2 pas -1
        saisieFact =  saisieFact * compteur
FIN
```

Pour cet exo, je me suis contentée de faire une traduction littérale de l'algo.

Allez, je teste.

Bigre !! Je n'arrive pas à afficher mon volet **Exo 33**! Dès que je clique sur l'onglet, je constate que c'est le volet **Exo 31** qui a systématiquement le focus. D'où cela peut-il bien venir ?

Ah !! Ca y est! C'est l'instruction **RepriseSaisie(saisieNbNotes)** qui se trouve dans **Initialisation de ongletExos** qui est responsable de ce phénomène.

Ce type de problème s'appelle un effet de bord : la façon dont on a développé les traitements associés au volet **Exo 31** a des conséquences sur le comportement du volet **Exo 33**.

Quand je clique sur le volet **Exo 33** de l'onglet, l'événement **A chaque modification de ongletExos** est exécuté. Dans cet événement, j'appelle l'événement **Initialisation de ongletExos**, et le focus est donc donné au champ **saisieNbNotes** du volet **Exo 31**.

Bon, il faut changer cette instruction de place.

Je la supprime du traitement **Initialisation de ongletExos**.

Maintenant, je la rajoute dans l'événement **A chaque modification de ongletExos**, mais je prends mes précautions. Je précise que je ne veux donner le focus au champ **saisieNbNotes** que si le volet d'onglet que l'utilisateur veut afficher (c'est-à-dire sur lequel il clique) est le volet **Exo 31**. Dans Windev, les volets d'onglets sont numérotés, et la propriété **Valeur** d'un onglet contient à tout instant le numéro de volet sélectionné par l'utilisateur.

Ici, le volet **Exo 31** porte le numéro 1, d'où le code suivant, à rajouter dans l'événement **A chaque modification de ongletExos** :

```
SI ongletExos..Valeur = 1 ALORS
    RepriseSaisie(saisieNbNotes)
FIN
```

Bon, maintenant, je peux tester si mon exo 33 fonctionne. Oui!

Exercice 34

Écrire l'algorithme qui calcule et affiche le nom et le salaire net de chacun des employés d'une entreprise et qui calcule et affiche également le salaire net moyen des employés.

Dans cette entreprise, les employés sont payés à l'heure (ils ont tous le même salaire horaire) et bénéficient d'une prime d'ancienneté.

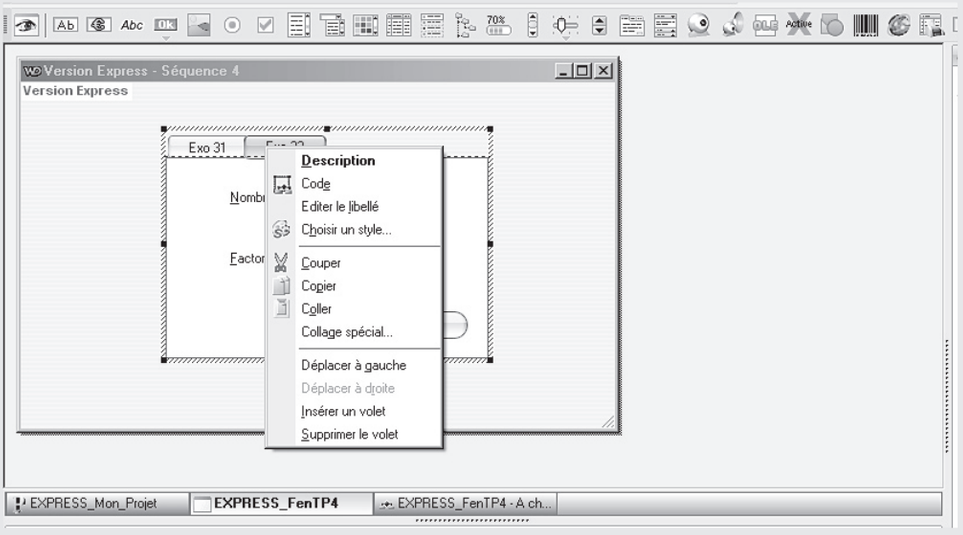
Le nombre n d'employés est saisi, ainsi que le salaire horaire et le taux de retenue.

Pour chaque employé, le nombre d'heures travaillées le montant de la prime et le nom de l'employé sont également saisis au clavier.

La règle de calcul d'un salaire net est la suivante : ((nombre d'heures travaillées * salaire horaire) + prime) * (1 – taux de retenue).

Vous pouvez faire cet exo, pour réinvestir ce qu'on a vu à l'exercice 31 : les saisies étant itératives, le problème est le même qu'à l'exercice 31, il faut éclater le code sur le modèle de l'exo 31.

Pour programmer cet exercice 34, commencez par rajouter un volet à notre onglet.



Pour rajouter un volet, faites un clic droit sur l'entête d'un des volets d'onglet (là où il est écrit **Exo 31** ou **Exo 33**). Dans le menu contextuel qui s'affiche au clic droit, choisissez **Insérer un volet**.

Ensuite, pour que ce nouveau volet soit le troisième de notre onglet, utilisez la commande **Déplacer à gauche** du menu contextuel.

Je voudrais juste profiter de cet exo pour vous faire utiliser une zone de liste permettant la sélection, autrement dit une **combo box**, appelée **combo** dans Windev.

Nous utiliserons cette combo pour y mettre les civilités concernant les employés.

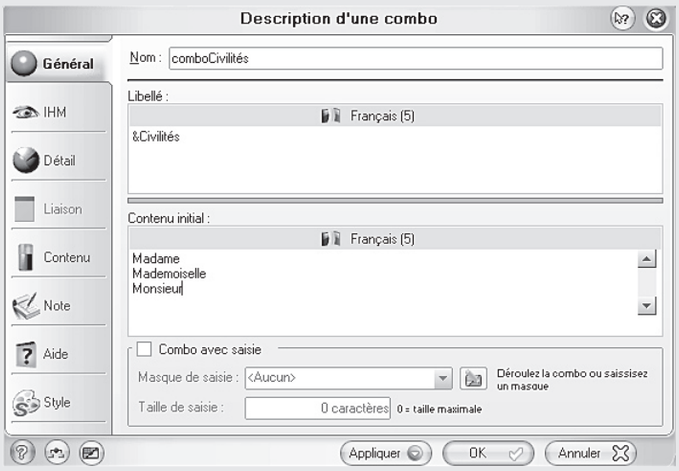
Monsieur, madame, mademoiselle : c'est ça des civilités.

Posez une **combo** sur le volet d'onglet destiné à l'exo 34.

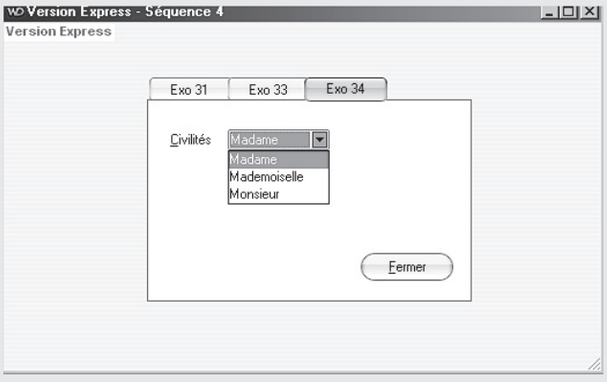
Lorsque vous posez une **combo**, windev vous demande comment vous souhaitez la remplir.

Laissez le choix proposé par défaut.

Donnez à notre **combo** les propriétés suivantes :



Voilà ce que cela donne à l'exécution :



Maintenant, ce qu'il nous faut savoir, c'est comment on utilise ce qui a été sélectionné par l'utilisateur.

Ce que je veux, c'est que le message indiquant le salaire affiche un truc du genre **Mademoiselle machin doit percevoir un salaire de ...** Il nous faut donc récupérer l'élément sélectionné et le concaténer au reste de notre texte.

Allez dans l'index de l'aide de Windev, tapez **combo**, puis choisissez **Manipuler une combo par programmation**. Dans la page d'aide qui s'affiche, descendez jusqu'à la rubrique **Récupérer l'élément sélectionné dans la combo.**

Voilà ci dessous ce qu'on nous explique...

Récupérer la valeur de l'élément sélectionné

Pour récupérer la valeur de l'élément sélectionné, utilisez une des syntaxes suivantes :

Syntaxe 1 (Combo sans saisie)

```
indice est un entier = <Nom Combo>
<Variable>=<Nom Combo>[indice]
```

Syntaxe 2 (combo sans saisie)

```
<Variable>=<Nom Combo>[<Nom Combo>]
```

Syntaxe 3 : Propriété Valeur affichée (combo avec ou sans saisie)

Code serveur uniquement

```
<Variable> = <Nom Combo>..ValeurAffichée
```

Bon, le plus simple pour nous, c'est la syntaxe 2.

Appliqué à notre combo, cette syntaxe 2 est elle qu'à tout instant **comboCivilités** contient l'élément de la combo qui est sélectionné.

Donc, en fait, c'est tout simple : si j'ai appelé **saisieNom** le champ de saisie du nom, pour afficher un message du genre **Mademoiselle machin doit percevoir un salaire de ...**, je dois écrire **combocivilités + saisieNom + "doit percevoir etc..."**.

Bon, je vous laisse terminer la programmation de cet exercice.

Une dernière remarque tout de même : vous avez peut-être remarqué que j'ai mis un accent au nom de notre combo **comboCivilités**. Windev est le seul langage de programmation que je connaisse et qui permette de mettre des accents dans les noms de variables et de composants graphiques. Donc, il vaut mieux éviter de prendre l'habitude de mettre des accents à nos variables et à nos objets graphiques.

Exercice 35

On laisse tomber, c'était l'algo qui était intéressant.

Exercice 36

Écrire l'algorithme du jeu suivant : ce jeu se joue à 2 joueurs.
Le premier joueur saisit un mot à l'abri du regard du deuxième joueur.
Puis, le deuxième joueur doit deviner quel est ce mot et le saisir.
Il a droit à 3 essais.
À chaque essai, le programme lui indique s'il a trouvé le mot ou bien, le cas échéant, quelles sont les bonnes lettres aux bonnes places parmi celles qu'il a saisies.

Exemple : si le mot à trouver est « carpette » et que le joueur a saisi « crateres », le programme lui affiche : c - - - e - - -.

Si, au bout du 3e essai, le joueur n'a pas trouvé le mot, un message lui indique quel était le mot à trouver.

Si on réfléchit bien, cet exercice comporte une saisie dans l'itération. Il faut donc éclater le code et le **pour** va devenir un **si**.

On a déjà programmé cet exercice dans le TP de la séquence précédente.

Les seuls traitements que l'on peut améliorer par rapport à la version de la séquence 3 sont ceux concernant le test du mot saisi caractère par caractère, où là, on peut remplacer le code existant par une itération.

Je vous propose donc de récupérer cette fenêtre et de l'adapter aux exigences de l'énoncé donné pour l'exo 36 séquence 4.

On ne va pas s'embêter à tout remettre sur un onglet, on va laisser la fenêtre telle quelle, la rajouter au projet TP séquence 4 et la modifier.

Notre projet contiendra donc finalement plusieurs fenêtres, il faudra donc prévoir un menu déroulant.

Pour rajouter une fenêtre déjà existante à un projet, Il y a deux méthodes. On peut dupliquer cette fenêtre, la mettre dans le répertoire du nouveau projet et l'ajouter au projet. On peut aussi tout simplement l'ajouter au projet : elle existera en un seul exemplaire et fera partie de deux projets.

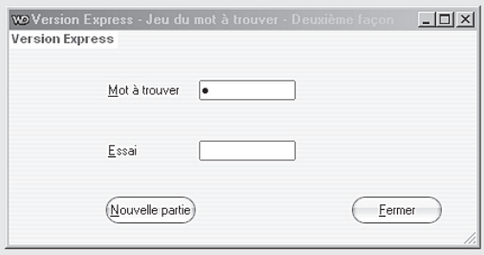
Pour cette fois, je retiendrai la première solution car nous allons modifier cette fenêtre et si on ne la duplique pas, elle n'existera plus dans son état antérieur.

Pour dupliquer une fenêtre sans sortir de Windev, il faut l'ouvrir en allant dans **Fichier** l **Ouvrir**. Il s'agit de la fenêtre que j'ai appelée **fenExo26Facon2.**

Ensuite, on va l'enregistrer sous un autre nom dans le répertoire de notre projet actuel en allant dans **Fichier** l **Enregistrer sous**, en sélectionnant le bon répertoire et en lui donnant un nom du style **fenExo36TP4**.

À ce moment là, windev nous demande si on veut rajouter la fenêtre au projet, on réponde **Oui**.

Voici cette fenêtre telle qu'elle était dans le TP de la séquence précédente…



Dans les codes de cette fenêtre, seul le code de l'événement **Sortie de saisieEssai** doit être modifié. On doit remplacer la partie de code suivante …

```
// Partie de code à modifier :
SI saisieEssai[[1]] <> saisieMotATrouver[[1]] ALORS
        saisieEssai = "-" + saisieEssai[[2]] + saisieEssai[[3]] + saisieEssai[[4]]
FIN
SI saisieEssai[[2]] <> saisieMotATrouver[[2]] ALORS
        saisieEssai = saisieEssai[[1]] + "-" + saisieEssai[[3]] + saisieEssai[[4]]
FIN
SI saisieEssai[[3]] <> saisieMotATrouver[[3]] ALORS
        saisieEssai = saisieEssai[[1]] + saisieEssai[[2]] + "-" + saisieEssai[[4]]
FIN
SI saisieEssai[[4]] <> saisieMotATrouver[[4]] ALORS
        saisieEssai = saisieEssai[[1]] + saisieEssai[[2]] + saisieEssai[[3]] + "-"
FIN
```

… par un traitement itératif, correspondant dans l'algo à :

```
Pour i de 1 à longueur(motATrouver)
Faire   Si      essai[i] <> motATrouver[i]
        Alors   essai[i] ← "-"
        FinSi
FinPour
```

La traduction du morceau d'algo ci-dessus en Windev va être plus compliquée car on ne peut pas écrire dans Windev l'équivalent de *essai[i]* ← "-", on est obligé de reconcaténer tous les caractères. Ca complique l'écriture du **pour**.

Alors voyons voir comment est-ce que c'est-il qu'on va t-on s'y prendre ?

En fait, on doit recopier **essai** sur lui-même sauf si le caractère courant (le i^e caractère) est différent du i^e caractère du mot à trouver. Dans ce dernier cas, on remplace le i^e caractère de **essai** par un tiret. Donc jusqu'à **i – 1**, on recopie **essai** sur lui-même, de **i + 1** à la fin du mot, on recopie aussi **essai** sur lui-même, et entre les 2, on remplace le i^e caractère de **essai** par un tiret, si celui-ci est différent du i^e caractère du mot à trouver.

Donc, ça donnerait quelque chose comme :

```
Pour i de 1 à longueur (MotATrouver)
Faire   Si      essai[i] <> MotATrouver[i]
        Alors   essai ← SousChaîne (essai, 1, i - 1) + "-" +
                SousChaîne (essai, i+1, longueur (MotATrouver))
        FinSi
FinPour
```

En ce qui concerne l'interface graphique de la fenêtre, il faut modifier la taille maxi de saisie des champs contenant le mot à trouver et les essais du second joueur, et le tour est joué.

Je vous laisse faire pour trouver le code correspondant au petit bout d'algo modifié ci-dessus. Si ce code vous donne du mal, c'est tant mieux, il est nécessaire que vous rencontriez des difficultés. Surtout, ne vous énervez pas, restez conscient(e) du fait que c'est « le métier qui rentre ».

Exercice 37 *(On laisse tomber, c'est la même chose que l'exo 31.)*

Écrire un algorithme qui affiche la valeur du plus grand de x nombres saisis, x étant saisi.

Exercice 38

Dans un but statistique, on veut établir des calculs de consommation d'un groupe de 100 abonnés à l'EDF, le dernier et l'avant-dernier relevé de compteur de chacun d'eux étant saisi.

On sait d'autre part, que la tarification se fait par tranches :

- **si la quantité d'électricité est inférieure à 100 kwh, le prix du kwh est de 0.09 euros ;**
- **si la quantité d'électricité est supérieure à 100 kwh, les 100 premiers kwh sont à 0.09 euros et au delà, à 0.05 euros.**

Le coût forfaitaire de location du compteur est de 6 euros hors taxes.

Ecrire l'algorithme qui calcule et affiche le pourcentage d'abonnés faisant partie de la première tranche, de la deuxième tranche, ainsi que la consommation moyenne de chaque tranche et du groupe complet d'abonnés.

Celui-là, faites-le, pour réinvestir ce qui a été vu précédemment : éclatement du code, création d'une variable globale pour le compteur d'itérations (dont je vous conseille de limiter la valeur à 5 environ et pas à 100 comme indiqué dans l'énoncé d'algo), affichage des constantes et des résultats dans des champs en lecture seule…

Exercice 39

Traduction des mots en code secret. On l'a tous fait quand on était petit!

Écrire l'algorithme qui traduit un mot de n'importe quelle longueur saisi au clavier en codes numériques ascii.

Exemple : si l'utilisateur saisit le mot jour, l'algorithme affiche le message suivant : « Le code secret du mot jour est 106 111 117 114 » (ces nombres étant les codes ascii des caractères j, o, u, et r).

Celui-là, il existe déjà, il n'y a qu'à le modifier. Pour cela, il faut récupérer la fenêtre qui est dans le TP séquence 2, la rajouter au projet du TP séquence 4 et modifier le code. Faites-le si le cœur vous en dit mais ne vous sentez pas obligé. Il n'y a qu'une saisie et l'itération démarre ensuite, donc, le mode de traduction est le même que pour l'exo sur le calcul du factoriel.

Exercice 40

Version 3 : cryptage d'un texte saisi par l'utilisateur. Les phrases sont cryptées dans l'ordre inverse où elles sont saisies dans la phrase.

Note : les points et les espaces ne font pas l'objet d'un cryptage particulier.

On va récupérer également cette fenêtre dans le TP de la séquence 3 et modifier le code de manière à ce qu'il corresponde à la version 3 de cet exo. Comme pour l'exo sur les factoriels et l'exo précédent, le code sera très proche de l'algo car il n'y a qu'une saisie à l'extérieur de l'itération.

Je tiens à ce que vous le fassiez car on va le reprendre plus loin pour le modifier à nouveau.

Utilisez la dernière version présentée dans le corrigé des algos : version 3 améliorée avec concaténation (ce que j'ai appelé version 4 dans le corrigé des exercices).

Attention, vous ne pourrez pas traduire tel quel cette partie de l'itération :

```
Si texte[i] = " " et texte[i+1] = "."
Alors     texte[i+1] ← " "
          Texte[i] ← "."
```

FinSi, mais vous devrez à la place utiliser la concaténation, comme on a fait dans le jeu du mot à trouver.

Exercice 41

Écrire un algorithme qui simule le jeu de pile ou face.

Déroulement du jeu : l'utilisateur saisit la lettre P pour pile, et F pour face, puis valide sa saisie (ou bien il clique sur le bouton « Pile » ou le bouton « Face » dans le cas d'une interface graphique et événementielle).

Le programme lui, choisit aléatoirement un nombre entre 0 et 1, si le nombre tiré au sort est 0, alors pile est gagnant, face est perdant, si le nombre tiré au sort est 1, alors pile est perdant et face est gagnant. Le programme affiche un message : gagné ou perdu.

L'utilisateur fait 10 essais et c'est au bout des 10 essais que la partie s'arrête, affichant son score à l'utilisateur : nombre de pertes et de gains.

Je choisis la dernière version figurant dans le corrigé des exercices de la séquence 4.

Ici, on est à nouveau dans un cas où il faut éclater le code car les saisies utilisateur sont dans l'itération. Récupérez la fenêtre correspondant à l'exo 22 séquence 3, rajoutez-la au projet, et réinvestissez ce qu'on a vu plus haut. On reprendra cette fenêtre plus loin pour, elle aussi, la modifier.

Exercice 42

Version 2 : modifier l'exo 42 pour qu'il sache compter les phrases, les mots et les lettres dans un texte saisi puis nettoyé au préalable (texte auquel on a enlevé tous les espaces inutiles).

Celui-là, c'est la première fois que vous avez à le faire, il n'en existe pas de version antérieure. La version avec le nettoyage est encore plus intéressante car elle vous oblige à passer d'écritures comme **texte[i]** ← à des écritures utilisant la concaténation.

En outre, le code est assez long par rapport aux code issus des autres algos, c'est intéressant d'apprendre à écrire des programmes de plus en plus longs.

Procédez progressivement : commencez par écrire et mettre au point la partie concernant le nettoyage, et seulement après, arrachez-vous les cheveux sur la partie concernant le comptage.

Exercice 43

On laisse tomber cet exo.

Exercice 44

Écrire l'algorithme qui affiche les tables de multiplication de 1 à 10.

Celui-là, on le laisse sauvagement tomber… pour le moment… on le retrouvera plus loin.

Exercice 45

On ne le fait pas.

Partie 2

La structure TantQue…FinTantQue

1. Présentation de la structure

Syntaxe Windev :

```
TANTQUE condition
                .. .
FIN
```

Dans Windev, le **TantQue** se comporte exactement comme décrit dans le cours d'algo.

Allez on attaque les exos !

Exercice 46 *(reprise de l'exo 31)*

Écrire l'algo de calcul et d'affichage de la moyenne d'un groupe d'élèves à un devoir, le nombre d'élèves étant saisi.

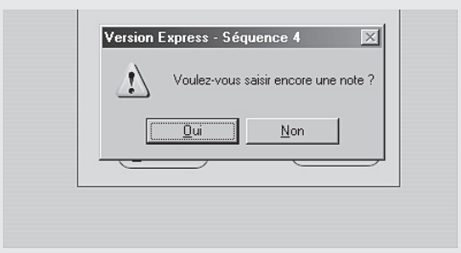
Il est inutile de vouloir traduire cet exo avec un **TANTQUE** : il fait partie de ceux pour lesquels on doit éclater le code, la saisie étant dans l'itération. Donc, les traductions de l'exo 31 et de l'exo 46 donnent exactement la même fenêtre.

Exercice 47 *(reprise de l'exo 31 avec modification de la condition d'arrêt)*

Écrire l'algo de calcul et d'affichage de la moyenne d'un groupe d'élèves à un devoir, l'algo demandant à l'utilisateur : "Saisir encore une note (o pour oui, n pour non) ?"

On va reprendre l'exo 31 et le modifier à nouveau. Pour le traduire, on va utiliser une nouvelle boîte de dialogue prédéfinie dans Windev : **OuiNon**.

Cette boîte de dialogue a l'allure suivante :



L'affichage de cette boîte de dialogue est le résultat de l'instruction **OuiNon (« Voulez-vous saisir encore une note ? »)**.

Cette instruction est un appel à la fonction **OuiNon**.

Cette fonction renvoie un booléen en guise de résultat : vrai si l'utilisateur a cliqué sur **Oui**, faux s'il a cliqué sur **Non**.

Voici le comportement du programme correspondant à l'exercice 47 : le volet d'onglet **Exo 47** s'affiche, permettant à l'utilisateur de saisie une note.

Lorsque l'utilisateur valide sa saisie, une boîte de dialogue s'ouvre, demandant à l'utilisateur s'il veut saisir encore une note.

Si l'utilisateur choisit **Oui**, on lui permet de saisir une note supplémentaire, sinon, on affiche le résultat du calcul.

Sur le volet d'onglet de l'exo 31, nous allons y modifier l'interface du volet d'onglet de l'exo 31, en supprimant le champ de saisie **Nombre de notes**.

Dans l'événement **A chaque modification de ongletExos**, remplacer l'instruction **RepriseSaisie(saisieNbNotes)** par l'instruction **RepriseSaisie (saisieNote")** pour donner le focus au champ de saisie des notes.

Dans l'événement **Sortie de saisieNote**, les instructions sont quasiment les mêmes que dans l'exo 31 sauf que l'ordre a un peu changé :

```
cumulNotesExo31 = cumulNotesExo31 + saisieNote
SI OuiNon("Voulez-vous saisir encore une note?") ALORS
        saisieNote = ""
        compteurExo31 = compteurExo31 + 1
        saisieNote..Libellé = "&Saisir "+ NumériqueVersChaîne(compteurExo31) + "° note :"
        RepriseSaisie (saisieNote)
SINON
        saisieMoyenne = cumulNotesExo31 / compteurExo31
        saisieMoyenne..Visible = Vrai
FIN
```

Fastoche ! Sauf que quand on teste, on consate que ça fonctionne mais notre volet **Exo 31** modifié pour l'exo 47 a encore un défaut : quand l'utilisateur clique sur **Réinitialiser**, le champ de saisie **saisieNote** n'obtient pas le focus. C'est parce que l'instruction qui redonne le focus à ce champ ne figure plus dans **Initialisation de ongletExos**, mais dans **A chaque modification de ongletExos**. Ce dernier événement ne se produit pas lorsqu'on reste sur le même volet d'onglet.

Donc, on **re-re-change de place** le bloc d'instructions

```
SI ongletExos..Valeur = 1 ALORS
        RepriseSaisie(saisieNote)
FIN
```

Que l'on met à la fin du code de l'événement **Initialisation de ongletExos**, et cette fois, le comportement convient.

Exercice 48 *(reprise exo 34)*

Modifiez l'exo 34 de manière à ce qu'il corresponde à l'algo de l'exo 48. Il y en a pour deux secondes. Il vous suffit de réinvestir ce qu'on vient de voir dans l'exo précédent.

Exercice 49 *(reprise de l'exo 35)*

Non, celui là, on ne le fait pas. Il y en a donc pour 0 seconde.

Exercice 50

Découverte du nombre magique.

Le but du jeu est de découvrir un nombre compris entre 1 et 100 choisi par l'ordinateur.

Déroulement du jeu :

L'utilisateur saisit un nombre. L'ordinateur lui indique si ce nombre est égal, plus petit ou plus grand que le nombre à découvrir.

En cas d'égalité, l'ordinateur donnera le nombre de coups qui ont été nécessaires pour découvrir ce nombre, l'utilisateur a droit à 5 coups au maximum.

Celui-là, on le fait.

Comme pour beaucoup des itérations précédentes, il faut éclater le code.

Il faut aussi lancer le traitement dans l'événement de sortie du champ où l'utilisateur saisit ses essais.

Si on regarde l'algo, les deux premières instructions que l'on voit sont **nbOrdi ← hasard(100)** et **nbCoups ← 0**.

Les 2 variables **nbOrdi** et **nbCoups** ne doivent être ni déclarées, ni initialisées dans l'événement de sortie du champ où l'utilisateur saisit ses essais successifs. Comme pour l'exercice 31, on doit donc déclarer ces variables comme étant des variables globales (dans la partie **Déclarations globales** de la fenêtre), et les initialiser dans le code d'initialisation de l'onglet.

Ainsi, dans la partie **Déclarations globales** de la fenêtre on aura quelque chose du genre :

```
// données pour exo31 (calcul de moyenne)
cumulNotesExo31 est un entier
compteurExo31 est un entier
// données pour exo 50 (nombre magique)
nbOrdiExo50 est un entier
nbCoupsExo50 est un entier
```

Voici maintenant le code de **Intialisation de ongletExos**

```
// exo 31
SI ongletExos..Valeur = 1 ALORS
        cumulNotesExo31 = 0
        compteurExo31 = 1
        saisieNote = ""
        saisieMoyenne = ""
        saisieMoyenne..Visible = Faux
        RepriseSaisie(saisieNote)
FIN
// exo 50
SI ongletExos..Valeur = 4 ALORS
        InitHasard
        nbOrdiExo50 =  Hasard (100)
        nbCoupsExo50 = 0
FIN
```

Vous pouvez constater que j'ai encadré d'un test les initialisations, pour chaque exercice (31 et 50), d'un test. Je ne provoque les initialisations que si le bon volet d'onglet est sélectionné par l'utilisateur.

Bien entendu, le traitement **Initialisation** se produit à l'ouverture, à un moment où l'utilisateur n'a encore sélectionné aucun volet d'onglet.

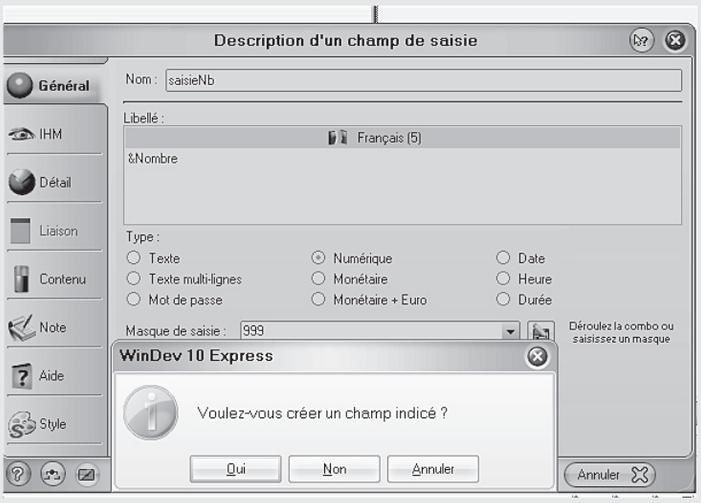
Alors pourquoi je fais ça ?

Parce que le traitement d'initialisation est appelé dans l'événement **A chaque modification** et que cet événement se produit lorsqu'il utilisateur sélectionne un autre volet d'onglet que le volet actuel. Ce traitement d'initialisation est également appelé dans l'événement **Clic sur boutonReinitialiser** de l'exercice 31.

Voilà, les cas de **nbOrdi** et **nbCoups** sont réglés.

La troisième instruction de l'algo est **nb ← -1**; **nb** est le champ de saisie où l'utilisateur saisit ses essais. Commençons donc par poser ce champ **saisieNb** sur l'onglet correspondant à l'exo 50.

Je prends un champ de saisie dans la barre de composants, je le pose sur le volet d'onglet de l'exo 50 et je l'appelle **saisieNb**, je clique sur **Numérique**, je choisis un masque de saisie de 3 chiffres, et voilà que tout à coup, s'affiche la boîte de dialogue suivante…



Ben qu'est-ce que c'est que ce truc, qu'est-ce qu'il a, qu'est-ce qu'il veut savoir au juste ?

Et en plus, si je clique sur **Non**, quand ensuite je valide la description de mon composant, il me remet le même message !

Je n'ai donc pas d'autre choix que de choisir la réponse **Oui** à cette question dont le sens m'échappe.

En fait, windev me propose cela car j'ai déjà un composant qui s'appelle **saisieNb** (sur le volet **Exo 33**). Si, dans une même fenêtre, j'appelle deux champs de saisie de la même façon, pour Windev, cela signifie que je veux créer un groupe de champs (dans notre cas, le groupe s'appellerait **saisieNb**), indicés signifiant que je pourrai manipuler tous les champs s'appelant **saisieNb** par des numéros (saisieNb[1] désignant le premier champ saisieNb, saisieNb[2] désignant le deuxième etc…). C'est puissant ça. Ca permet d'écrire un seul traitement et de l'appliquer à tout un groupe de champs. On essaiera de s'en souvenir pour un utilisation ultérieure. On pourrait même s'en servir dans ce TP pour s'entraîner. On pourrait utiliser l'exo des tables de multiplication pour se faire la main.

Toujours est-il que pour notre exo 50, on ne veut pas créer un groupe de champs s'appelant **saisieNb**, on veut juste pouvoir poser un champ de saisie sur l'onglet de l'exo 50. Je crois qu'on n'a pas d'autre choix que de lui changer son nom, à ce champ de saisie. Voyons voir… Comment va-t'on l'appeler… **saisieNombreMagique**, oui, c'est pas mal ça, et au moins, on est sûr qu'il n'y en a pas deux qui s'appelleront comme ça.

On aurait pu aussi appeler **saisieNbExo33** le champ de saisie du volet d'onglet de l'exercice 33, et **saisieNbExo55** celui qui nous préoccupe maintenant, mais finalement, **saisieNombreMagique**, c'est mieux!!

Bon maintenant que ce petit problème de nom est réglé, je dois me demander ce que devient l'instruction **nb ← -1**, ou plutôt **saisieNombreMagique ← -1**.
Rappelez-vous : dans l'algo, on avait décidé de mettre cette valeur bidon dans saisie**NombreMagique** pour "forcer" l'entrée dans le traitement itératif.

En fait, étant donné qu'on a l'intention de mettre le traitement dans l'événement de sortie de **saisieNombreMagique**, le traitement sera forcément lancé uniquement si l'utilisateur a saisi un essai. Donc, cette instruction n'a plus lieu d'être.

Occupons-nous maintenant du traitement proprement dit.

La condition **TantQue nb <> nbOrdi et nb <= 5 Faire** se transforme en un **Si…Alors**.

Les instructions **Afficher ("Entrer votre nombre ")** et **Saisir (nb)** disparaissent.

Toutes les autres instructions **Afficher** deviennent des messages **Info**.

Essayez d'écrire seul(e) le code de l'événement sortie de champ **saisieNombreMagique**.

Si vraiment vous avez du mal, regardez ci-dessous, mais pas avant d'avoir essayé tout(e) seul(e) pendant au moins une heure car il est temps que vous deveniez de plus en plus autonomes.

Voilà un début de solution :

```
SI saisieNombreMagique <> nbOrdiExo50 ET nbCoupsExo50 <= 5 ALORS
        SI saisieNombreMagique > nbOrdiExo50 ALORS
                Info ("Votre nombre est trop grand")
        SINON
                SI saisieNombreMagique <  nbOrdiExo50  ALORS
                        Info ("Votre nombre est trop petit")
                FIN
        FIN
        nbCoupsExo50 ++
FIN
SI saisieNombreMagique = nbOrdiExo50 ALORS
        Info ("Bravo, vous avez gagné en ", nbCoupsExo50, "coups")
SINON
        Info ("Dommage, le nombre choisi était :", nbOrdiExo50)
FIN
```

Mais ça ne marche pas comme on voudrait : testez vous-même et vous constaterez que d'abord, l'onglet de l'exo31 nous casse régulièrement les pieds avec son message **Voulez-vous saisir encore une note** et que l'utilisateur se voit afficher le message **Dommage, le nombre choisi était…** après avoir saisi son premier essai.

Il faut donc faire quelques petites mises au point car notre produit est invendable en l'état.

Réglons déjà le problème de notre pauvre utilisateur qui se fait jeter comme un malotru.

Si je trace la fenêtre, je me rends compte que quelque soit le nombre d'essais, on rentre dans la partie **SI saisieNombremagique = nbOrdi**, et ça, il ne faut pas, donc…donc…on va peut-être bien mettre un **SINON** au lieu de mettre un deuxième **SI**. Ce qui donne :

```
SI saisieNombreMagique <> nbOrdiExo50 ET nbCoupsExo50 <= 5 ALORS
        SI saisieNombreMagique > nbOrdiExo50 ALORS
                Info("Votre nombre est trop grand")
        SINON
                SI saisieNombreMagique <  nbOrdiExo50  ALORS
                        Info("Votre nombre est trop petit")
                FIN
        FIN
        nbCoupsExo50 ++
SINON
        SI saisieNombreMagique = nbOrdiExo50 ALORS
                Info ("Bravo, vous avez gagné en ", nbCoupsExo50, "coups")
        SINON
                Info ("Dommage, le nombre choisi était :", nbOrdiExo50)
        FIN

FIN
```

Si on teste, on constate qu'on a réglé le problème, enfin… on l'a tellement bien réglé que pour ma part, j'ai droit à 6 essais au lieu de 5 !!

Bon, ça, c'est facile à corriger. Je remplace
SI saisieNombreMagique <> nbOrdiExo50 ET nbCoupsExo50 < = 5 ALORS

Par
SI saisieNombreMagique <> nbOrdiExo50 ET nbCoupsExo50 < 5 ALORS

Occupons-nous maintenant du message issu du volet **Exo31** de l'onglet. Si on teste un peu plus, on se rend compte que l'utilisateur, lorsqu'il est sur le volet On rencontre ce genre de problème parce qu'on n'a pas créé de fenêtres séparées pour chaque exercice mais des plans d'onglet. Si on va fouiller dans l'aide, on découvre en lisant que **lors du changement d'un onglet, le code de sortie du champ qui a le focus est activé**.

Donc, lorsque l'utilisateur clique sur un autre volet d'onglet que le volet actif, l'événement de sortie du champ qui a le focus est exécuté, et l'événement **A chaque modification** de l'onglet est exécuté seulement après.
Voilà, notre malheur vient de là. Il faut qu'on trouve le moyen d'empêcher l'exécution du code de sortie du champ qui a le focus.

Il faut qu'on s'arrange pour que les deux zones de saisie responsables de nos malheurs (**saisieNote** et **saisieNombreMagique**) soient vides à l'ouverture de la fenêtre ou au clic sur l'onglet concerné, et on n'exécute l'événement de sortie de ces zones de saisie que si elles ne sont pas vides.

J'ai une idée que j'essaie : j'encadre le code **Sortie de saisieNote** à l'aide du test **SI saisieNote <> "" ALORS…FIN**, et je fais pareil pour le code **Sortie de saisieNombreMagique**, pour lequel j'écris **SI saisieNombreMagique <> "" ALORS…FIN**.

Je teste. Le problème n'est pas réglé, les événements de sortie des champs sont exécutés comme si je n'avais pas rajouté ces tests.

Je trace le projet (en allant dans **Projet** l **Mode test**, et en choisissant **Tracer le projet**). En traçant, je me rends alors compte que le traitement entre dans le **SI saisieNote <> ""** et dans le **SI saisieNombreMagique <> ""**, comme si ces champs n'étaient pas vides.

Si j'affiche la valeur des champs à l'aide d'une instruction **info**, je me rends compte qu'ils ne sont pas vides, puisqu'ils valent **0** (zéro). Voilà l'origine de notre problème.

Donc, ces tests ne conviennent pas.

Finalement, qu'est-ce qu'on veut exactement ? Ce qu'on veut, c'est que quand l'utilisateur passe du volet d'onglet **Exo 31** ou **Exo 50** à un autre, l'événement de sortie des champs de saisie **saisieNote** et **saisieNombreMagique** ne soient pas exécutés.

Je vais dans l'aide de windev, et je fourrage à droite à gauche.

Je découvre que les champs de saisie possèdent une propriété qui s'appelle **Modifié**, vraie si l'utilisateur a modifié un champ (par saisie au clavier ou à la souris).

Alors j'encadre le code **sortie de saisieNote** avec **SI saisieNote..Modifié ALORS** et **FIN**, et j'encadre le code **Sortie de saisieNombreMagique** avec **SI saisieNombreMagique..Modifié ALORS** et **FIN**.

Une fois qu'on a fait ces mises au point, le comportement de la fenêtre est correct.

Je teste à nouveau le jeu du nombre magique.

Il fonctionne, l'utilisateur a bien droit à 5 essais, mais il reste deux petites améliorations à apporter au comportement de ce volet d'onglet :

- redonner le focus au champ **saisieNombreMagique** après affichage de chaque message d'information **Votre nombre est trop… (grand ou petit)** ;
- prévoir un bouton **Rejouer** qui réinitialise le volet d'onglet **Exo 50**.

Je vous laisse faire.

Exercice 51

Algo PV : Si un conducteur a entre 0.5 et 0.8 gramme d'alcool par litre de sang, il a un retrait de permis de 6 mois et une amende de 200 euros par décigramme au dessus de 0.5 gramme, 0.5 gramme inclus.

Si le conducteur a entre 0.8 et 1 gramme d'alcool dans le sang, il a un retrait de permis de 24 mois et une amende de 300 euros par décigramme au dessus de 0.5 gramme, 0.5 gramme inclus.

Au delà de 1 gramme, le conducteur repasse son permis et paye une amende de 5 000 euros.

Un litre d'alcool pur pèse 800 grammes. Un homme adulte possède environ 8 litres de sang. Une bouteille d'alcool à 45° possède 45% d'alcool.

À titre indicatif, un verre de 10 cl de boisson alcoolisée à 10° donne une alcoolémie de 0.1 g par litre de sang.

Écrire un algorithme qui calcule l'amende à payer et le temps de retrait de permis, la quantité de boisson et la teneur en alcool de chaque boisson étant saisies. L'utilisateur indique en saisissant le nombre 0 qu'il ne désire plus saisir de nouvelle boisson.

À faire.

Bien entendu, on ne va pas demander à l'utilisateur de saisir 0 s'il ne veut plus saisir de boisson, mais on va lui poser la question à l'aide d'une boîte de dialogue **OuiNon**. Je vous rappelle que cet exo existe déjà dans la séquence précédente et que vous n'avez qu'à rajouter la fenêtre au présent projet et la modifier. C'est l'affaire de quelques millisecondes. Éclatez le code sur le modèle des exercices précédents.

Exercice 52 *(reprise de l'exo 37)*

Hop là, celui-là, on le saute allègrement.

Exercice 53 *(reprise de l'exo 36)*

Écrire l'algorithme du jeu suivant : ce jeu se joue à 2 joueurs.

Le premier joueur saisit un mot à l'abri du regard du deuxième joueur. Puis, le deuxième joueur doit deviner quel est ce mot et le saisir. Il a droit à un nombre d'essais fixé par le premier joueur. A chaque essai, le programme indique au joueur s'il a trouvé le mot ou bien, le cas échéant, quelles sont les bonnes lettres aux bonnes places parmi celles qu'il a saisies.

Exemple : si le mot à trouver est « carpette » et que le joueur a saisi « cratères », le programme lui affiche : c - - - e - - -.

Si, au bout du nombre d'essais autorisés, le joueur n'a pas trouvé le mot, un message lui indique quel était le mot à trouver.

Si vous en avez le temps, l'envie, le courage, allez récupérer l'exo 36 de cette séquence et modifiez-le pour qu'il satisfasse aux exigences de l'énoncé. Vous n'avez pas besoin d'aide pour cela.

Exercice 54 *(Reprise de l'exo 40 sauf que …)*

Votre client est agent secret et écrit ses messages en langage crypté. Il vous demande de lui écrire un programme qui crypte le texte qu'il saisit. Ce cryptage consiste à inverser le mot et à intercaler entre chacune de ses lettres la lettre qui a la position symétrique dans l'alphabet. Je vous explique : l'alphabet comporte 26 lettres : a b c d e f g h i j k l m n o p q r s t u v w x y z. Dans l'alphabet a est la symétrique de z, b est la symétrique de y, c est la symétrique de x,..., et ainsi de suite jusqu'à m qui est la symétrique de n.

Celui-là, vous pouvez le faire si vous voulez : la difficulté était dans l'algo mais vous pouvez remodifier l'exo 40 de ce TP pour voir ce que ça donne une fois programmé.

Exercice 55 *(alors là, accrochez-vous, personnellement je l'adore celui-là)*

Écrire l'algo qui affiche les déplacements d'une balle de ping-pong qui rebondit sur les bords d'un carré, dont la taille est choisie par l'utilisateur.

La balle part d'une position aléatoire au bas du carré. La direction que prend la balle après avoir rebondi est le résultat d'un choix aléatoire entre gauche ou droite.

L'algo s'arrête quand la balle a rebondi 40 fois.

La balle est représentée par un petit bouton dont on change la position par programmation. On suppose que la taille du bouton est de 1 sur 1.

On suppose également qu'il existe pour les objets graphiques les propriétés « colonne » et "ligne" qui contiennent les coordonnées de l'objet par rapport au bord de la fenêtre sur laquelle il est posé, ces coordonnées pouvant se modifier par programmation.

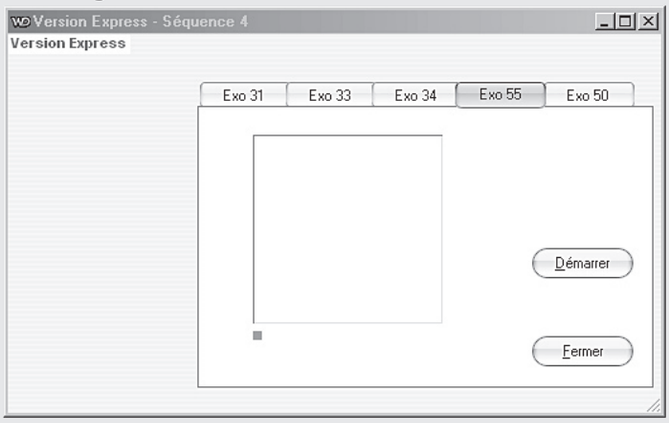
Pour celui-là, je vous donne un coup de main. Il faut **absolument** que vous le programmiez. Le code est très proche de l'algo car il n'y a aucune saisie dans l'itération.

Les propriétés **colonne** et **ligne** utilisées dans l'algo s'appellent respectivement **X** et **Y** dans Windev (quelle chance hein…).

Dans l'aide de windev, on nous fournit l'explication suivante à propos de la propriété **X**, qui est l'**abscisse** du champ **:** Si le champ appartient à une fenêtre, cette abscisse est exprimée en pixels. Elle correspond à la position horizontale du coin haut gauche du champ par rapport au coin haut gauche de la zone client de la fenêtre (c'est-à-dire la fenêtre sans les barres de titre et de menu ni le cadre).

Et voici l'explication qui nous est fournie à propos de la propriété **Y**, qui est l'**ordonnée** du champ : dans une fenêtre, cette ordonnée est exprimée en pixels. Cette ordonnée correspond à la position verticale du coin haut gauche du champ par rapport au coin haut gauche de la zone client de la fenêtre (c'est-à-dire la fenêtre sans les barres de titre et de menu ni le cadre).

On va créer le volet d'onglet suivant :



Voici ci-dessous la description de chacun des composants de ce volet d'onglet **Exo55**. Dans le tableau ci-dessous, je vous indique le nom du composant, sa nature (bouton, libellé…), sur quel onglet de la fenêtre **Description** vous trouverez la propriété à modifier, le nom de la propriété à modifier et la valeur que vous devez lui donner.

Nom du composant	Nature du composant	Onglet	Nom de la propriété	Valeur
cadre	Champ libellé	Général	Libellé	Rien
		IHM	Largeur	150
			Hauteur	150
		Style	Elément	Cadre extérieur
			Cadre	Creux
balle	Champ libellé	Général	Libellé	Rien
		IHM	Largeur	7
			Hauteur	7
		Style	Couleur fond	Orange foncé
boutonDemarrer	Bouton	Général	Libellé	Démarrer

Maintenant, le code. Essayez dans un premier temps de traduire seul(e) l'algorithme et testez votre programme. Mettez-le au point en étant autonome le plus possible. Seulement si vous êtes en panne, regardez ci-dessous.

Voilà pour le moment la traduction littérale de l'algo :

```
Clic sur boutonDémarrer
versdroite, vershaut sont des booléens
i est un entier
InitHasard
// par rapport à l'algo j'ai changé droite et haut en versdroite et vershaut
versdroite = (Hasard (0,1) = 0)
vershaut = Vrai
balle..X = Hasard (cadre..X, cadre..X + cadre..Largeur)
balle..Y = cadre..Y + cadre..Hauteur-1
POUR i = 1 A 40
        // Attention, la condition composée ci-dessous doit être
        // écrite sur une seule ligne
        TANTQUE (balle..X >= cadre..X + 1)
                ET (balle..X <= cadre..X - 1 + cadre..Largeur)
                ET (balle..Y >= cadre..Y + 1)
                ET (balle..Y <= cadre..Y + cadre..Hauteur - 1)
        // fin de ce qui doit être écrit sur une seule ligne
        SI versdroite ALORS
                        balle..X++
                SINON
                        balle..X--
                FIN
                SI vershaut ALORS
                        balle..Y--
                SINON
                        balle..Y++
                FIN
        FIN
        SI modulo (i, 2) = 0 ALORS
                vershaut = PAS vershaut
        SINON
                versdroite = PAS versdroite
        FIN
FIN
```

Une fois que vous avez tapé ce code, si vous le testez, il ne se passe pas grand chose : on voit la balle se déplacer, mais seulement une fois. Elle ne fait qu'un trajet. Et pourquoi donc ?

Essayez de trouver pourquoi en exécutant le code pas à pas : vous avez vu ? on n'entre dans le **TANTQUE** que pour **i = 1**, dès que i passe à 2, les conditions d'entrées ne sont plus satisfaites, et c'est la raison pour laquelle la balle ne fait qu'un trajet sur les 40 prévus.

Notre balle va trop loin au premier trajet et sa position au début du second trajet dépasse les bornes autorisées.
Cherchez une solution pour régler ce problème. Si vous n'en trouvez pas, je vous en propose deux ci-dessous, une un peu « cracra » et l'autre plus futée.

La première solution (la soluce « cracra ») : elle consiste à faire « reculer » la balle d'un cran (en fonction de la position précédente) à la fin du trajet pour pouvoir entrer dans le **TANTQUE** du trajet suivant.

Voici le code :

```
                    // …Je ne vous remets pas le code depuis le début
                    SI vershaut ALORS
                              balle..Y--
                    SINON
                              balle..Y++
                    FIN
          // Le mot clé FIN qui suit est le FIN de notre TANTQUE
          FIN

          // si on ne fait pas ce qui suit, on ne peut pas entrer dans l'exécution
          // du trajet suivant
          SI versdroite ALORS
                    balle..X--
          SINON
                    balle..X++
          FIN
          SI vershaut ALORS
                    balle..Y++
          SINON
                    balle..Y--
          FIN
          // fin de ce qu'il faut rajouter pour entrer dans le trajet suivant
          SI modulo (i, 2) = 0 ALORS
                    vershaut = PAS vershaut
          SINON
                    versdroite = PAS versdroite
          FIN
FIN
```

Testez. Ca fonctionne, la balle fait ses 40 trajets. Par contre, il reste plein de traces des trajets précédents, ce qui nous fait plein de taches orange autour de notre créé. C'est pas beau.

D'où ce phénomène vient-il ? Il vient du fait que notre programme mobilise les ressources système sans lui rendre la main. Du coup, le système ne peut pas rafraîchir correctement l'affichage.

Nous allons rajouter l'instruction **multitâche(0)**, juste avant le **FIN** du **TANTQUE**; cette instruction a pour effet de rendre la main au système.

On teste. Ça fonctionne : notre balle rebondit sur les côtés du cadre, et il n'y a plus aucune trace orange qui traîne.

Nous allons essayer d'obtenir le même résultat avec la **seconde solution**, ma préférée.

En quoi consiste cette **seconde solution ?**

On crée un nouveau booléen **changedir**, **vrai** si on vient de changer de direction.

On entre dans le **TANTQUE** si les bornes ne sont pas dépassées **ou** bien si **changedir** est **vrai** (c'est-à-dire si on vient de changer de direction).

Dès qu'on est entré dans le **TANTQUE**, on fait passer **changedir** à **faux** car on ne vient plus de changer de direction et c'est alors la valeur des bornes qui agira pour l'arrêt des itérations.

Une fois qu'on est sorti du **TANTQUE**, on remet **changedir** à **vrai**, afin de pouvoir entrer dans le **TANTQUE** suivant.

Voilà le code, j'ai mis en gras tous les ajouts concernant **changedir** :

```
versdroite, vershaut, changedir sont des booléens
i est un entier
InitHasard
// par rapport à l'algo j'ai changé droite et haut en versdroite et vershaut
versdroite = (Hasard (0,1) = 0)
vershaut = Vrai
balle..X = Hasard (cadre..X, cadre..X + cadre..Largeur)
balle..Y = cadre..Y + cadre..Hauteur-1
changedir = Vrai
POUR i = 1 A 40
        // Attention, la condition composée ci-dessous doit être écrite sur une seule ligne
        TANTQUE (balle..X >= cadre..X + 1) ET (balle..X <= cadre..X - 1 +
                cadre..Largeur) ET (balle..Y >= cadre..Y + 1) ET (balle..Y <= cadre..Y +
                cadre..Hauteur - 1) OU changedir
                changedir = Faux
                SI versdroite ALORS
                        balle..X++
                SINON
                        balle..X--
                FIN
                SI vershaut ALORS
                        balle..Y--
                SINON
                        balle..Y++
                FIN
                Multitâche(0)
        FIN

        SI modulo (i, 2) = 0 ALORS
                vershaut = PAS vershaut
        SINON
                versdroite = PAS versdroite
        FIN
        changedir =Vrai
FIN
```

Voilà. Testez. Vous constatez que ça fonctionne tout aussi bien qu'avec la solution précédente. Je préfère cette dernière solution car elle relève plus d'une réelle réflexion, alors que la solution précédente, bien qu'efficace, relève plus de ce qu'on appelle la bidouille.

Cet exercice peut vous servir de base pour créer tout un tas de jeux comme par exemple la casse brique, le tennis etc…

Passons maintenant à la dernière partie de ce TP.

Partie 3

La structure Répéter…jusqu'à

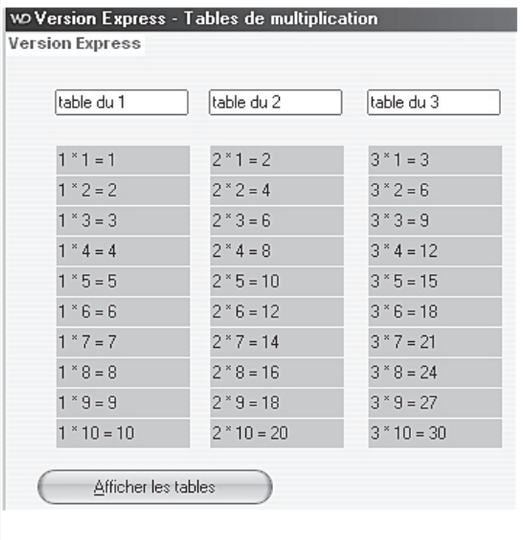
1. Présentation de la structure

Il n'existe pas tout à fait l'équivalent du **répéter** en Windev, cependant, avec la syntaxe suivante, on peut reproduire le comportement d'une structure **répéter**.

```
Boucle
        // instructions
Si condition alors Sortir
Fin
```

On va juste dans cette partie faire 1 exo : l'exo 44, celui avec les tables de multiplication. Il faut d'abord traduire le **pour** en **répéter** imbriqués puis en Windev avec l'instruction **boucle**.

Faire cet exo a pour moi un double intérêt : vous faire tester l'instruction **boucle** et vous faire manipuler des composants **indicés** (vous vous rappelez, on en a parlé un peu plus haut dans ce TP). Le code restera très proche de l'algo car c'est un traitement itératif ne nécessitant aucune saisie. Voici l'allure de notre fenêtre …

À l'ouverture…	Quand l'utilisateur a cliqué sur le bouton **Afficher les tables**
	

et ainsi de suite pour les autres tables….

Je vais vous guider pour créer cette fenêtre.

Posez sur la fenêtre un champ de saisie, appelez le **ligne**, il représentera une ligne d'une table. Ce champ de saisie n'a pas de libellé.

Faites un clic-droit sur ce champ et dans le menu contextuel, choisissez **Copier**, faites un clic droit n'importe-où ailleurs sur la fenêtre et choisissez **Coller**, recommencez jusqu'à ce que vous ayez 10 champs de saisie identiques sur la fenêtre.

Allez dans la description de chacun des champs, appelez les tous **ligne**. À partir du deuxième champ que vous décidez d'appeler **ligne**, windev vous demande si vous voulez créer un champ indicé, répondez oui.

Ainsi, la première zone de saisie de cette colonne sera accessible par le nom **ligne[1]**, la seconde par le nom **ligne[2]** et ainsi de suite.

Dans cette première série de champs de saisie, on va afficher la table du 1, il faut ensuite une série de champs pour la table du 2, une pour la table du 3 etc.

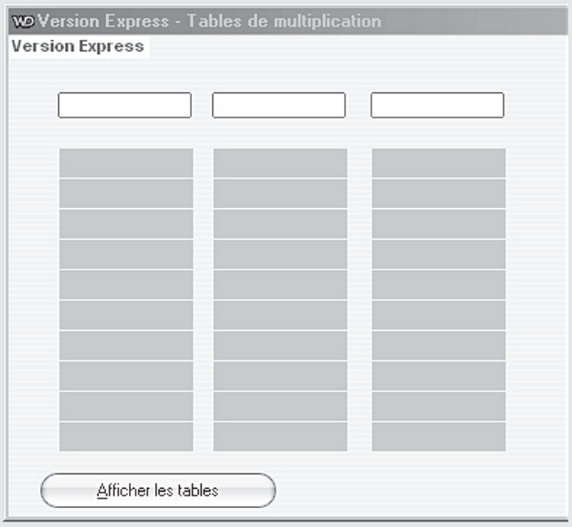
Il suffit de créer une deuxième colonne de 10 champs, de les appeler aussi **ligne**, et de continuer ainsi jusqu'à de que vous ayez 10 paquets de 10 champs sur votre fenêtre.

Les champs de saisie de la seconde colonne s'appelleront **ligne[11]**, **ligne[12]…ligne[20]**, ceux de la troisième colonne seront indicés de 21 à 30 et ainsi de suite, jusqu'à a dernière colonne dont les champs **ligne** seront indicés de 91 à 100.

Maintenant, on va créer une autre famille de champs indicés, celle où vont s'afficher les titres **table du 1**, **table du 2** etc.

Posez un nouveau champ de saisie sur votre fenêtre et appelez-le **titre**. Donnez-lui le style qui vous plaît puis copiez et collez-le 10 fois, en créant ainsi 10 champs indicés s'appelant **titre**. Rajoutez un bouton **Afficher les tables**.

Voilà. L'interface de la fenêtre est terminée.



Occupons-nous du code maintenant.

Commencez par traduire l'algo **Tables de multiplication** avec une structure **répéter**. Si vous êtes en difficulté, je vous ai mis la solution ci-dessous.

Lexique	Algorithme tables du fois avec pour	Algorithme tables du fois avec répéter
i, j (entiers, calculés) : compteurs d'itérations.	Début Pour i de 1 à 10 Faire Afficher ("Table du ", i) Pour j de 1 à 10 Faire Afficher (i, " * ", j, " = ", i * j) FinPour FinPour Fin	Début i ←1 Répéter Afficher("Table du ", i) j ←1 Répéter Afficher (i, " * ", j, " = ", i * j) j ← j + 1 Jusqu'à j =10 i ← i + 1 Jusqu'à i = 10 Fin

Nous voulons traduire l'algo en Windev et ce que nous voulons, c'est que l'instruction **Afficher (i, " * ", j, " = ", i * j)** soit traduite de manière à ce que l'affichage s'effectue dans nos champs de saisie **ligne**, de même que nous voulons que **Afficher("Table du ", i)** soit traduite de manière à ce que l'affichage s'effectue dans nos champs **titre**.

Nous avons sur notre fenêtre 100 champs **ligne**, indicés de 1 à 100.

L'expression **ligne[1]** désigne le premier champ **ligne** en haut à gauche de la fenêtre, *ligne[2]* le deuxième champ en haut à gauche etc. Le principe de numérotation est le même pour

les champs indicés **titre**, dont les noms vont de **titre[1]** (pour le titre de la première colonne, celle de la table du 1), à **titre[10]**.

Réfléchissons !!!

Dans les champs de saisie **titre**, on veut écrire **Table du 1**, **Table du 2** etc.

Donc, on écrira l'instruction **titre[i]= "Table du " + i** qui, dans l'algo correspond à **Afficher ("Table du ", i).**

Ca c'est facile, car la valeur du compteur correspond exactement à l'information que l'on veut afficher.

Pour traduire **Afficher (i, " * ", j, " = ", i * j)**, par contre, c'est plus compliqué. On voit bien sur la copie d'écran ci-dessus que les indices des champs de saisie **ligne** n'ont pas un rapport direct avec la valeur des compteurs **i** et **j**.

On va se faire un petit tableau qui a pour but d'étudier la valeur de l'indice du champ de saisie en fonction de la valeur de i et de j, dans l'espoir de trouver une règle de calcul. Dans les cases du tableau, il y a la valeur de l'indice du champ de saisie où doit s'afficher le résultat de la multiplication :

		Valeurs de i		
		1	2	3
Valeurs de j	1	1	11	21
	2	2	12	22
	3	3	13	23
	4	4	14	24
	5	5	15	25
	6	6	16	26
	7	7	17	27
	8	8	18	28
	9	9	19	29
	10	10	20	30

Quand i vaut 1(table du 1), l'indice du champ de saisie est égal à j.

Quand i vaut 2, l'indice du champ de saisie est égal à j + 10.

Quand i vaut 3, l'indice du champ de saisie est égal à j + 20.

Ce qu'il faut, c'est trouver une règle de calcul de l'indice du champ de saisie en fonction de i et de j.

Cherchez une bonne quinzaine de minutes avant de regarder ci-dessous la solution.

L'indice du champ de saisie où on doit afficher i * j est **10 * (i - 1) + j**.

Par conséquent, l'instruction correspondant à **Afficher (i, " * ", j, " = ", i * j)** est **ligne[10*(i-1)+j] = i + " * " + j + " = " + i*j**.

Vous savez tout, alors allez-y, écrivez le code dans le clic du bouton **Affiche les tables**.

Si vraiment vous n'y arrivez pas, je vous ai mis le code ci-dessous.

```
i, j, indice sont des entiers
i = 1
boucle
        titre[i] = "table du " + i
        j = 1
        boucle
                ligne[10 * (i - 1) + j] =  i + " * " + j + " = " + i * j
                j++
                si j > 10 alors
                        sortir
                FIN
        FIN
        i++
        si i > 10 alors
                sortir
        FIN
FIN
```

Bien. Nous avons fait tous les exos utiles à faire.

Maintenant, rappelez-vous, je voulais que vous appreniez à faire un vrai fichier d'aide Windows avec sommaire, index et recherche.

Windev fournit des outils permettant de créer une aide utilisateur.

Allez voir dans l'index de l'aide de windev, à la rubrique **Aide d'une application**, vous trouverez tout ce dont vous avez besoin.

Bon, ce TP sur les itérations est terminé.