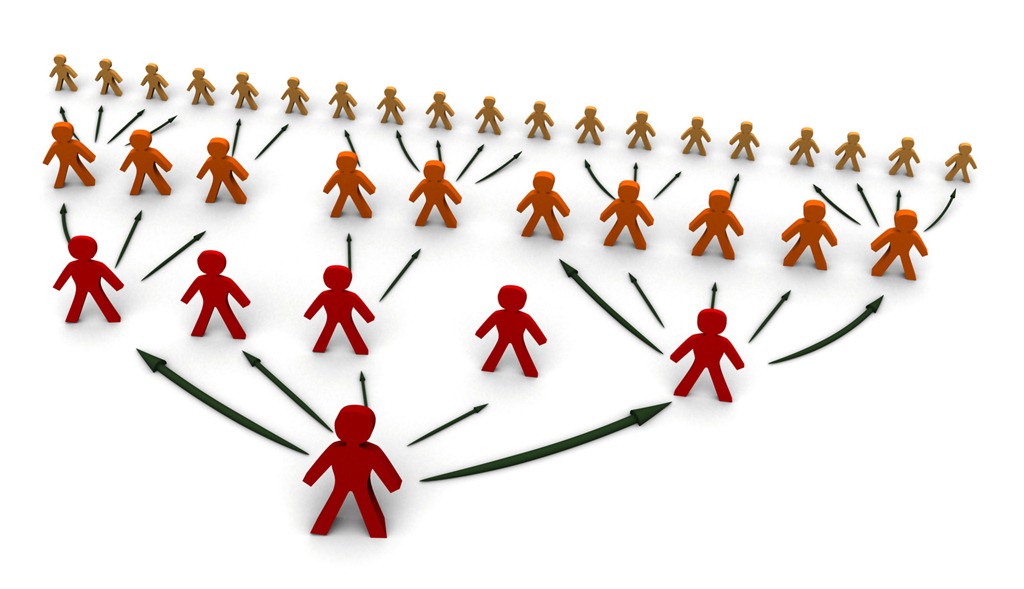
Vente multiniveau



|  |  |
| --- | --- |
| Auteur | Dwenn Kaufmann |
| Enseignant | Jean-Paul Wettstein |
| Date de fin de projet | 14/03/14 |
| Module | TIN\_AP1 |
| Version | 1.0 |

Table des matières

[1 Introduction 4](#_Toc382530054)

[2 Documentation de développement 4](#_Toc382530055)

[2.1 Explications détaillées du projet 4](#_Toc382530056)

[2.2 Diagramme des cas d'utilisation 5](#_Toc382530057)

[2.3 Arborescence des fichiers 5](#_Toc382530058)

[2.4 Définition des conventions applicables 6](#_Toc382530059)

[3 Réalisation des cas d'utilisation 7](#_Toc382530060)

[3.1 Ajouter une personne 7](#_Toc382530061)

[3.1.1 Scénario 7](#_Toc382530062)

[3.1.2 Maquettes 7](#_Toc382530063)

[3.1.3 La phase de programmation 7](#_Toc382530064)

[3.1.4 La phase de tests 8](#_Toc382530065)

[3.2 Afficher les détails d’une personne 8](#_Toc382530066)

[3.2.1 Scénario 8](#_Toc382530067)

[3.2.2 Maquettes 9](#_Toc382530068)

[3.2.3 La phase de programmation 9](#_Toc382530069)

[3.2.3.1 ToolTyp 9](#_Toc382530070)

[3.2.3.2 MessageBox 9](#_Toc382530071)

[3.2.3.3 Classe Personne 10](#_Toc382530072)

[3.2.4 La phase de tests 10](#_Toc382530073)

[3.3 Modifier une personne 10](#_Toc382530074)

[3.3.1 Scénario 10](#_Toc382530075)

[3.3.2 Maquette 11](#_Toc382530076)

[3.3.3 La phase de programmation 11](#_Toc382530077)

[3.3.3.1 frmPrincipal 11](#_Toc382530078)

[3.3.3.2 frmPersonne 11](#_Toc382530079)

[3.3.4 La phase de tests 13](#_Toc382530080)

[3.4 Supprimer une personne 14](#_Toc382530081)

[3.4.1 Scénario 14](#_Toc382530082)

[3.4.2 Maquette 14](#_Toc382530083)

[3.4.3 La phase de programmation 14](#_Toc382530084)

[3.4.4 La phase de tests 15](#_Toc382530085)

[3.5 Parcourir l’arbre 15](#_Toc382530086)

[3.5.1 Scénario 15](#_Toc382530087)

[3.5.2 La phase de programmation 15](#_Toc382530088)

[3.5.3 La phase de tests 16](#_Toc382530089)

[3.6 (Re)calculer les informations d’une personne 16](#_Toc382530090)

[3.6.1 Scénario 16](#_Toc382530091)

[3.6.2 La phase de programmation 17](#_Toc382530092)

[3.6.2.1 frmPrincipal 17](#_Toc382530093)

[*3.6.2.2* *Classe Personne* 17](#_Toc382530094)

[3.6.3 La phase de tests 18](#_Toc382530095)

[3.7 Sauver l’arbre 18](#_Toc382530096)

[3.7.1 Scénario 18](#_Toc382530097)

[3.7.2 La phase de programmation 19](#_Toc382530098)

[3.7.3 La phase de tests 19](#_Toc382530099)

[3.8 Ouvrir un arbre existant 19](#_Toc382530100)

[3.8.1 Scénario 19](#_Toc382530101)

[3.8.2 La phase de programmation 19](#_Toc382530102)

[3.8.3 La phase de tests 21](#_Toc382530103)

[4 Problèmes rencontrés 22](#_Toc382530104)

[4.1 Impossibilité de modifier les Treenodes une fois un arbre ouvert 22](#_Toc382530105)

[4.1.1 Description 22](#_Toc382530106)

[4.1.2 Solution 22](#_Toc382530107)

[4.2 Conversions en double requérant une virgule comme délimiteur 22](#_Toc382530108)

[4.2.1 Description 22](#_Toc382530109)

[4.2.2 Solution 22](#_Toc382530110)

[5 Conclusions 23](#_Toc382530111)

[6 Annexes 23](#_Toc382530112)

[6.1 Cahier des charges 23](#_Toc382530113)

[6.2 Code source commenté 23](#_Toc382530114)

[6.2.1 frmPrincipal.cs 23](#_Toc382530115)

[6.2.2 frmPersonne.cs 28](#_Toc382530116)

[6.2.3 Personne.cs 30](#_Toc382530117)

[6.3 Références 33](#_Toc382530118)

# Introduction

Le but de ce travail est de proposer à l’utilisateur de visualiser une vente à multiniveau à l’aide une application réalisée en groupe.

L’environnement de développement utilisé est Microsoft Visual Studio 2013.

Le langage utilisé est le C#, langage dont nous avons appris les bases lors de notre CFC, et avons obtenus d’avantage de théorie sur celui-ci ici au Cifom.

Dans ce rapport vous trouverez :

* Une documentation de développement :
  + Une explication détaillée du projet,
  + Les conventions de nommage
* Les cas d’utilisation divers de l’application, comprenant pour chaque cas :
  + Un diagramme
  + Une explication des points clés du code
  + Les différents tests effectués afin de s’assurer des résultats escomptés
* Les problèmes rencontrés ainsi que leurs solutions
* Annexes diverses :
  + Cahier des charges
  + Référence
  + Code source.

# Documentation de développement

## Explications détaillées du projet

Le but du projet est de présenter à l’utilisateur une vue graphique d’un arbre de vente multiniveau. Tout partira d’un point de départ, le « Parent » appelé ‘moi’, (l’utilisateur), possède plusieurs caractéristiques, modifiables qui sont :

* Nom
* Volume personnel
* Remarque

Il possède également plusieurs caractéristiques, qui elles, seront (re)calculées à chaque modification apportée à n’importe quel endroit de l’arbre, les voici :

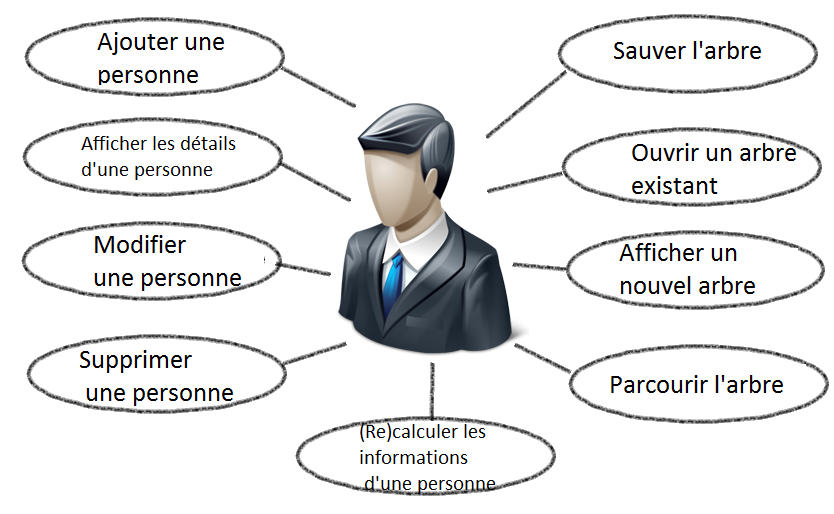
* Niveau
* Volume de groupe
* Gain Théorique
* Gain Net
* Gain Associés

Cette personne de base peut avoir des personnes « enfants », des personnes donc que la personne originelle aura « rapporté » dans l’arbre en les recrutant par exemple. Ces mêmes personnes pourront elles-aussi avoir, comme leur parent, des enfants. Ceci permet de recréer la topologie d’un arbre de vente multiniveau.

Le principe final de l’application sera, à l’aide d’une vue graphique des différentes personnes insérées, de visualiser le gain de chaque personne de l’arbre avec son pourcentage de bonus, obtenu grâce au niveau.

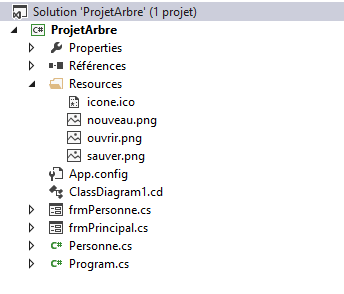
La répartition exacte des différents gains entre les personnes seront expliqués plus tard dans ce rapport.

## Diagramme des cas d'utilisation



## Arborescence des fichiers

Voici l’arborescence des fichiers tels que l’on peut les voir depuis l’environnement de développement :



## Définition des conventions applicables

Pour notre application, nous avons choisi une convention de nommage, discutée au sein du groupe, qui nous a permis de nous retrouver facilement avec chaque variable.

Dans les Formulaires :

* Les variables commencent par un préfixe qui correspond à son type :
  + Bool : bVariable,
  + Char : cVariable,
  + DialogResult : drVariable,
  + Double : dblVariable,
  + Int  : iVariable,
  + OpenFileDialog : ofdVariable,
  + SaveFileDialog : sfdVariable,
  + Stream : sVariable,
  + String : strVariable,
  + TreeNode : tnVariable.
* Les contrôles ont eux aussi un préfixe qui désigne leur type :
  + Bouton : btnContrôle,
  + Formulaire : frmFormulaire,
  + Label : lblContrôle,
  + MenuStrip : msContrôle,
  + RichTextBox : rtbxContrôle,
  + Textbox : tbxContrôle,
  + ToolStripMenuItem : tsmContrôle,
  + TreeView : tvContrôle.
* Personne :

Le cas des personnes est un peu exceptionnel, car dans le corps du programme, nous avons deux types de personnes bien définis. Il y a la personne nommée « moi » qui est la personne tout en haut de l’arbre. Depuis cette variable il est possible de faire des modifications directes. Puis on trouve aussi la personne P, elle est utilisée pour modifier la personne sur laquelle on est situé dans la l’arbre.

* Au sein de la classe personne :

Pour la classe, nous avons choisi de laisser les attributs sans préfixe pour que l’accès depuis le code soit plus simple et, comme cela, il y a une certaine concordance avec les propriétés. Mais pour tout ce qui touche aux variables utilisés dans les méthodes, nous avons quand même mis les préfixes.

# Réalisation des cas d'utilisation

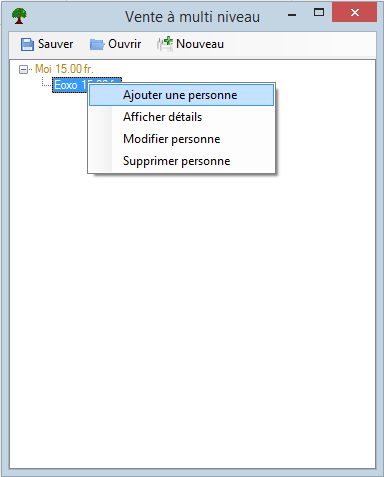
## Ajouter une personne

### Scénario

Ce cas nous sert à ajouter une personne dans notre TreeView, en tant qu’enfant de la personne sur laquelle on effectue un clic droit, afin de créer la vue graphique des enfants sous une forme d’arbre. Cette arborescence sera modifiable en tout temps sur le programme.

Cette action s’effectue soit en faisant un clic droit sur une personne, soit en double cliquant dessus.

### Maquettes



### La phase de programmation

private void ajouterPersonneToolStripMenuItem\_Click(object sender, EventArgs e)

{

P = new Personne();

N = new TreeNode();

N.Tag = P;

N.Text = P.Nom + " " + P.VolumePersonnel.ToString("c");

N.ToolTipText = P.Affiche(true);

N.ContextMenuStrip = contextMenuStrip1;

N.ForeColor = P.Couleur;

treeView1.SelectedNode.Nodes.Add(N);

treeView1.SelectedNode.Expand();

Parcourir(treeView1.Nodes[0]); //On y envoie le début du Treeview

}

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Clic droit/double clic plus bas ou plus haut qu’une personne | Ne fait rien | OK |  |

## Afficher les détails d’une personne

### Scénario

Cette action sert à afficher les caractéristiques d’une personne contenue dans le TreeView.

Il y a deux méthodes pour afficher ceux-ci :

* Survoler une personne avec le pointeur de la souris, dans ce cas-ci il est possible de voir toutes les informations concernant la personne, sauf les remarques éventuelles.
* Faire un clic droit sur la personne, et dans ce cas, une fenêtre apparaîtra, contenant cette fois-ci les informations complètes sur la personne.

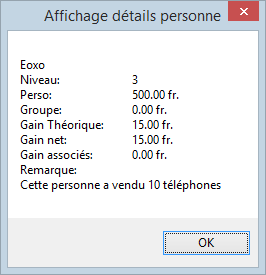
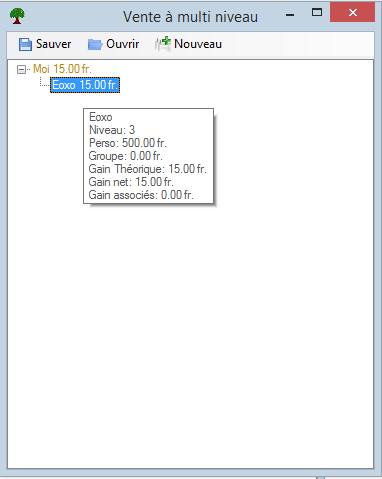
Il existe cette différence car cela était demandé dans le cahier des charges.

Nous avons donc pensé à deux moyens différents de régler le problème :

* Créer une méthode supplémentaire dans la classe personne (mais non conforme au diagramme de classe fourni dans le cahier des charges)
* Modifier la méthode d’affichage des paramètres, en lui passant un paramètre suivant si l’on désire ou non qu’elle nous renvoie la remarque.

La deuxième solution étant plus adaptée, nous avons choisi de passer un paramètre de type Booléen, afin de déterminer si oui ou nous désirions recevoir la remarque concernant la personne ou non.

### Maquettes



### La phase de programmation

#### ToolTyp

tnCourant.ToolTipText = P.Affiche(true);

#### MessageBox

private void afficherDetailsStripMenuItem\_Click(object sender, EventArgs e)

{

P = (Personne)(tvPrincipal.SelectedNode.Tag);

MessageBox.Show(P.Affiche(false), "Affichage détails personne");

}

#### Classe Personne

public string Affiche(bool bTooltip)

{

if (bTooltip == true) //Si c'est pour le tooltiptext, la remarque ne doit pas être présente.

{

return nom + "\nNiveau: " + niveau + "\nPerso: " + volumePersonnel.ToString("c") + "\nGroupe: " + volumeGroupe.ToString("c") + "\nGain Théorique: " + gainTheorique.ToString("c") + "\nGain net: " + gain.ToString("c") + "\nGain associés: " + gainAssocies.ToString("c");

}

else

{

return nom + "\nNiveau:\t\t" + niveau + "\nPerso:\t\t" + volumePersonnel.ToString("c") + "\nGroupe:\t\t" + volumeGroupe.ToString("c") + "\nGain Théorique:\t" + gainTheorique.ToString("c") + "\nGain net:\t\t" + gain.ToString("c") + "\nGain associés:\t" + gainAssocies.ToString("c") + "\nRemarque:\n" + remarque;

}

}

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Tester si la remarque ne figure pas quand le tooltip d’une personne est affiché | La remarque n’y figure pas | OK. |  |
|  | Tester si la remarque est présente lorsque la fenêtre « Affichage des détails d’une personne » s’affiche | La remarque est présente | OK |  |

## Modifier une personne

### Scénario

Pour la modification nous avons créé un dialogue entre deux formulaires :

* frmPrincipal : celui qui nous affiche l’arbre et où l’on choisit la personne à modifier.
* frmPersonne : depuis où l’on a la possibilité de modifier les valeurs de la personne.

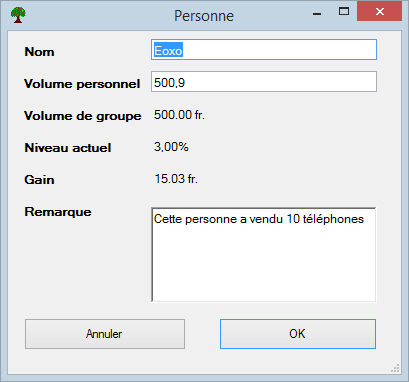
Le problème de ce formulaire a été de rendre les TextBox de saisies sensibles à toute erreur pouvant générer un problème dans le programme par la suite. Les deux problèmes étaient les suivants :

* Les noms pouvaient contenir n’importe quel caractère,
* Puis dans la TextBox du montant de vente de la personne il fallait gérer le fait que l’on entre que des nombres et un point ou une virgule.

Pour corriger ces erreurs, nous avons choisi d’utiliser des expressions régulières qui nous permettent de faire des tests assez poussés pour les zones de saisie.

Ensuite, il a fallu gérer la pression du bouton « OK » et du bouton « ANNULER » pour que les variables soient bien assignées dans la personne « P » choisie dans notre arbre.

### Maquette



### La phase de programmation

#### frmPrincipal

private void modifierPersonneToolStripMenuItem\_Click(object sender, EventArgs e)

{

P = (Personne)(tvPrincipal.SelectedNode.Tag); //Passage du TAG dans la personne

frmPersonne mod = new frmPersonne(P); //Création Form Dialogue

if (mod.ShowDialog() == DialogResult.OK)//Test si pression "OK" si oui attribuation de la personne modifier

{

P = mod.Pmod;

P.CalculNiveau();

tvPrincipal.SelectedNode.Tag = P;

tvPrincipal.SelectedNode.ToolTipText = P.Affiche(true);

tvPrincipal.SelectedNode.ForeColor = P.Couleur;

}

Parcourir(tvPrincipal.Nodes[0]);

}

#### frmPersonne

public Personne Pmod;

bool bValideVol = true;

bool bValideNom = true;

public frmPersonne(Personne P)

{

InitializeComponent();

AcceptButton = btnOk; //A la pression du bouton "Enter", va simuler un clic sur la touche Ok.

Pmod = P;

tbxNom.Text = P.Nom;

tbxVolumePersonnel.Text = Convert.ToString(P.VolumePersonnel);

lblVolumeGroupeValue.Text = P.VolumeGroupe.ToString("c");

double dblNiveau = P.Niveau / 100; //Variable utilisé suite a l'erreur d'affichage sur le formulaire de modification Car affichais 300,00% au lieux de 3,00%

lblNiveauActuelValue.Text = dblNiveau.ToString("p");

lblGainValue.Text = P.Gain.ToString("c");

rtbxRemarque.Text = P.Remarque;

}

private void button1\_Click(object sender, EventArgs e)

{

char cA = ',';

char cB = '.';

tbxVolumePersonnel.Text = tbxVolumePersonnel.Text.Replace(cB, cA); //Permet de transformer les "." en "," pour que ça soit valide pour les ocnverison futures du double.

double dblVol = Convert.ToDouble(tbxVolumePersonnel.Text);

dblVol = ((int)((dblVol \* 20) + 0.5));//Arrondie de la monnaie au 5ct

dblVol /= 20;

Pmod.Remarque = rtbxRemarque.Text;

Pmod.Nom = tbxNom.Text;

Pmod.VolumePersonnel = dblVol;

}

private void tbxVolumePersonnel\_Leave(object sender, EventArgs e)

{

if (Regex.IsMatch(tbxVolumePersonnel.Text, @"^[0-9]\*(?:[\, , \.][0-9]?[0-9])?$") && tbxVolumePersonnel.Text!="")

{

bValideVol = true;

tbxVolumePersonnel.ForeColor = Color.Black;

if (bValideNom == true && bValideVol == true)

{

btnOk.Enabled = true;

}

else

{

btnOk.Enabled = false;

}

}

else

{

bValideVol = false;

tbxVolumePersonnel.ForeColor = Color.Red;

btnOk.Enabled = false;

}

}

private void tbxNom\_TextChanged(object sender, EventArgs e)

{

if (Regex.IsMatch(tbxNom.Text, @"^[A-Z,a-z, ,\-,.]\*?$") && tbxNom.Text != "")

{

bValideNom = true;

tbxNom.ForeColor = Color.Black;

if (bValideNom == true && bValideVol == true)

{

btnOk.Enabled = true;

}

else

{

btnOk.Enabled = false;

}

}

else

{

bValideNom = false;

tbxNom.ForeColor = Color.Red;

btnOk.Enabled = false;

}

}

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Entrer un nom non conforme. | Le texte dans le champ doit devenir rouge et rendre impossible la pression du bouton OK | OK |  |
| Entre une valeur pour le volume personnel non conforme. | Le texte dans le champ doit devenir rouge et rendre impossible la pression du bouton OK | OK |  |
| Pression du bouton « Annuler » | Le formulaire doit se fermer sans faire aucune modification | OK |  |
| Pression bouton « OK » | Le formulaire doit se fermer puis insérer toutes les valeurs modifiées dans la personne puis refaire les calculs. | OK |  |

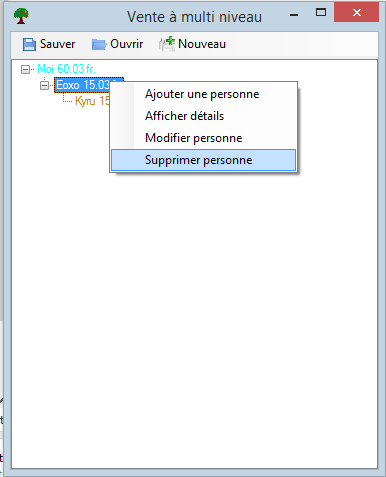
## Supprimer une personne

### Scénario

A première vue, ce cas d’utilisation peut paraître simpliste, en effet, chaque TreeNode possède une fonction Remove(). Le souci est qu’il faut que les enfants du TreeNode supprimé soient conservés, et réattribués au parent du TreeNode supprimé. Nous avons buté un certain moment sur le problème mais avons finalement trouvé une solution viable-

Notre solution est donc d’utiliser une boucle qui clone chaque node enfant de la node à supprimer, chez la node parent actuel de la node à supprimer. Ensuite, la node à supprimer est finalement supprimée. La node sélectionnée est maintenant celle qui était la node parent de la node à supprimer.

### Maquette



### La phase de programmation

private void supprimerPersonneToolStripMenuItem\_Click(object sender, EventArgs e)

{

TreeNode PositionApres = tvPrincipal.SelectedNode.Parent; //Permet de se repositionner sur la bonne TreeNode après avoir supprimé la node à Supprimer.

if (tvPrincipal.SelectedNode.Level != 0) //Permet d'éviter la suppression du Noeud initial. A chaque sous-noeud, incrémente de 1. Donc empêche la supression du noeud père.

{

TreeNode tnClone = (TreeNode)tvPrincipal.SelectedNode.Clone();

for (int i = 0; i < tnClone.Nodes.Count; i++) //Parcours toutes les nodes du niveau en cours

{

tvPrincipal.SelectedNode.Parent.Nodes.Add(tnClone.Nodes[i]); //Ajoute toutes les notes du niveau en cours au Parent, ce qui va les mettre au meme niveau que les nodes du parent de celles supprimées.

}

tvPrincipal.SelectedNode.Remove(); //Finalement, supprime la node de base qui a servi de modèle à la copie.

tvPrincipal.SelectedNode = PositionApres;

}

tvPrincipal.ExpandAll();

Parcourir(tvPrincipal.Nodes[0]);

}

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Supprimer un certain nombre de personnes afin de voir si les données du programme sont toujours cohérentes. | Nodes enfants affectés au parent de la node à supprimer | OK |  |
|  | Recalcul des propriétés de chaque treenode | Chaque treenode voit ses propriétés monétaires recalculées | OK |  |

## Parcourir l’arbre

### Scénario

C’est cette fonctionnalité qui va parcourir l’arbre en partant du premier TreeNode, et parcourir tous les enfants afin de permettre de tout recalculer lorsqu’il y a des changements de paramètres dans un ou plusieurs TreeNode.

Le principe de la fonction est de créer deux nodes :

* Parent,
* Enfant.

C’est grâce au tag de ces deux nodes que l’on pourra recréer les personnes afin de prendre les données à chaque utilisateur.

Cette méthode nous permet aussi d’assigner à chaque personne son volume de groupe et son gain associé. Deux variables qui nous sont utiles dans le calcul du gain net de la personne.

### La phase de programmation

private double Parcourir(TreeNode tnParent)

{

double dblTotal = 0;

double dblGainAss = 0; //Gain Associeé de l'enfant

//P = (Personne)(Parent.Tag); Ancienne ligne qui posait problème

Personne P = (Personne)(tnParent.Tag); //Création d'une autre personne car fonctionne mais le nom reste identique.

foreach (TreeNode tnEnfant in tnParent.Nodes)

{

dblTotal += Parcourir(tnEnfant);

Personne P2 = (Personne)(tnEnfant.Tag); //Création d'une autre personne car fonctionne mais le nom reste identique.

P2.CalculNiveau();

dblGainAss += P2.GainTheorique;

}

P.VolumeGroupe = dblTotal;

P.GainAssocies = dblGainAss;

dblTotal += P.VolumePersonnel;

P.CalculNiveau();

tnParent.Tag = P;

tnParent.Text = P.Nom + " " + (P.Gain).ToString("c");

tnParent.ForeColor = P.Couleur;

tnParent.ToolTipText = P.Affiche(true);

return dblTotal;

}

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Faire des appels de la fonction avec des valeurs différentes et analyser tous les nodes un après l’autre pour savoir si les valeurs sont correct partout. | Recalcule partout juste. | Le résultat est bon. |  |

## (Re)calculer les informations d’une personne

### Scénario

Ce cas d’utilisation est clairement ce qui nous a mis le plus de temps à développer, de par sa complexité à le comprendre, et ensuite, à l’implémenter. C’est ici que vont se faire tous les calculs relatifs aux propriétés monétaire d’une personne, c’est grâce à cette fonction que nous allons savoir exactement qui gagne combien.

La base de tous les calculs démarre avec le chiffre d’affaire de chaque personne. Ce chiffre d’affaire est obtenu grâce à une addition en son volume de vente personnel et son volume de groupe. Nous calculerons ensuite le niveau de la personne, niveau qui va permettre de déterminer le pourcentage de commission que la personne aura.

Vu que c’est un système de vente à multi niveau et non pyramidal (et donc légal), il paraît donc logique que le père de ses enfant ne gagnera pas la commission de tous ses enfants. Voici un récapitulatif permettant de mieux visualiser les choses.

* Chiffre d’affaire :
  + Chiffre d’affaire = volume personnel + volume du groupe
* Le gain théorique :
  + Le gain théorique = chiffre d’affaire \* pourcentage /100
* Le gain associé :
  + Le gain associé = gain théorique des enfants directs
* Le gain net :
  + Le gain net : gain théorique – gain associé

Maintenant un tableau qui va mieux faire comprendre la répartition du pourcentage :

|  |  |  |  |
| --- | --- | --- | --- |
| Chiffre d'affaire | | Pourcentage | Couleur |
| Min | Max |
| 0 | 500 | 1% | Black |
| 500 | 1’500 | 3% | DarkGoldenrod |
| 1’500 | 2’500 | 6% | Cyan |
| 2’500 | 4’800 | 9% | Firebrick |
| 4’800 | 6’500 | 12% | Maroon |
| 6’500 | 10’000 | 15% | Salmon |
| 10’000 | 20’000 | 18% | Green |
| 20’000 | ∞ | 20% | Red |

### La phase de programmation

#### frmPrincipal

P.CalculNiveau();

#### Classe Personne

public void CalculNiveau()

{

double dblChiffre = VolumeGroupe + VolumePersonnel;

//Si aucune de ces catégories, le niveau retourne à zero.

if (dblChiffre < 500)

{

Niveau = 1;

Couleur = Color.Black;

}

else if (dblChiffre >= 500 && dblChiffre < 1500)

{

Niveau = 3;

Couleur = Color.DarkGoldenrod;

}

else if (dblChiffre >= 1500 && dblChiffre < 2500)

{

Niveau = 6;

Couleur = Color.Cyan;

}

else if (dblChiffre >= 2500 && dblChiffre < 4800)

{

Niveau = 9;

Couleur = Color.Firebrick; //Ancien yellow mais la couleur ne se lisait pas bien

}

else if (dblChiffre >= 4800 && dblChiffre < 6500)

{

Niveau = 12;

Couleur = Color.Maroon;

}

else if (dblChiffre >= 6500 && dblChiffre < 10000)

{

Niveau = 15;

Couleur = Color.Salmon;

}

else if (dblChiffre >= 10000 && dblChiffre < 20000)

{

Niveau = 18;

Couleur = Color.Green;

}

else

{

Niveau = 20;

Couleur = Color.Red;

}

GainTheorique = dblChiffre \* Niveau /100;

Gain = GainTheorique - GainAssocies;

}

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Les valeurs obtenues doivent être correctes et répondre aux exigences du cahier des charges. | Valeurs correctes. | OK | Nous avons contrôlé avec M J-P. Wettstein que les valeurs de sorties soient correctes. |

## Sauver l’arbre

### Scénario

L’enregistrement du projet est un point qui nous a semblé critique. Il est également conseillé de le faire dans le cahier des charges. Nous avons utilisé une méthode permettant de Sérialser l’arbre en entier dans un fichier texte. L’enregistrement peut être fait de deux façons, Enregistrer(rapidement) et Enregistrer Sous. Nous avons pris cette idée sur le Notepad de Windows, qui permet qu’une fois que l’utilisateur aie sauvegardé au moins une fois, il suffise de presser CTRL+S ou de cliquer sur Enregistrer pour réenregistrer sur le même fichier sans confirmation.

Si l’utilisateur n’a encore jamais sauvegardé son arbre actuel ou ne désire pas l’enregistrer sous le même nom, un clic sur le bouton « Enregistrer sous… », va permettre de choisir le chemin ainsi que le nom du fichier à sauvegarder. Si une sauvegarde a déjà eu lieu, « Enregistrer sous… » va reprendre le nom du dernier enregistrement dans la fenêtre permettant de sauvegarder, et ceci automatiquement.

L’enregistrement peut être appelé à différents moments dans le programme :

* Pression du bouton enregistrer,
* Raccourci clavier standard Windows : CTRL + S,
* Lorsque l’on veut générer un nouveau projet sur la fenêtre actuelle,
* Ouverture d’un ancien projet déjà sauvegardé,
* Puis à la fermeture du programme.

Pour les trois derniers cas, une alerte sous forme de message demandera à l’utilisateur s’il souhaite enregistrer son arbre actuel avant de poursuivre, ce qui pourrait mener à la destruction non désirée d’un arbre sauvegardé au préalablement.

### La phase de programmation

private void Sauvegarder()

{

SaveFileDialog sfdEnregistrer = new SaveFileDialog();

sfdEnregistrer.DefaultExt = "txt";

sfdEnregistrer.Filter = "Fichier Text (\*.txt)|\*.txt";

sfdEnregistrer.FileName = strChemin;

sfdEnregistrer.ShowDialog();

if (sfdEnregistrer.FileName.Length > 0)

{

using (Stream sFile = File.Open(sfdEnregistrer.FileName, FileMode.Create))

{

BinaryFormatter bf = new BinaryFormatter();

bf.Serialize(sFile, tvPrincipal.Nodes.Cast<TreeNode>().ToList());

}

strChemin = sfdEnregistrer.FileName;

}

}

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Multiple sauvegarde du même projet afin de comparer si tous les fichiers sont identiques. | Les fichiers enregistrés sont identiques. | OK |  |
|  | Réenregistrer par-dessus un fichier déjà existant | Seul le dernier enregistrement persiste. | OK |  |

## Ouvrir un arbre existant

### Scénario

Une fois un arbre sauvegardé, il paraît logique de pouvoir le rouvrir. Nous avons donc rajouté cette fonctionnalité à l’application.

### La phase de programmation

private void Ouvrir()

{

try

{

OpenFileDialog ofdOuvrir = new OpenFileDialog();

ofdOuvrir.DefaultExt = "txt";

ofdOuvrir.Filter = "Fichier Text (\*.txt)|\*.txt";

ofdOuvrir.ShowDialog();

if (ofdOuvrir.FileName.Length > 0)

{

using (Stream sFile = File.Open(ofdOuvrir.FileName, FileMode.Open))

{

tvPrincipal.SelectedNode = tvPrincipal.Nodes[0];

tvPrincipal.SelectedNode.Remove();

BinaryFormatter bf = new BinaryFormatter();

object obj = bf.Deserialize(sFile);

TreeNode[] nodeList = (obj as IEnumerable<TreeNode>).ToArray();

tvPrincipal.Nodes.AddRange(nodeList);

strChemin = ofdOuvrir.FileName;

}

tvPrincipal.ExpandAll();

}

}

catch

{

MessageBox.Show("Problème lors du chargement du fichier, veuillez réessayer", "Erreur");

Application.Restart();

Ouvrir();

}

}

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Modification des TreeNode une fois un projet ouvert. | Permet de remodifier les propriétés des personnes une fois un nouvel arbre Ouvert. | OK |  |

# Problèmes rencontrés

## Impossibilité de modifier les Treenodes une fois un arbre ouvert

### Description

Après avoir au préalable sauvegardé un arbre, il nous était possible de le recharger et de l’affiche, mais la modification des Treenodes entraînait une erreur. En regardant de la documentation sur internet, nous avons pu découvrir d’où vient exactement le problème. La classe Personne, créée par nous-même, n’était pas sérialisable de base, ce qui fait que lors de la sérialisation du TreeView, seules les informations liées au TreeView pouvaient être sauvegardées, ainsi l’objet Personne contenu dans la propriété « Tag » de chaque TreeNode n’était jamais sauvegardé, ce qui entraînait la conséquence du plantage de l’application lors de la modification d’un TreeNode, car celle-ci utilise l’objet contenu dans la propriété « Tag » du TreeNode.

### Solution

Pour régler ce problème, il faut rajouter deux lignes dans la classe personne afin qu’elle puisse aussi bénéficier de la sérialisation.

Les lignes à rajouter dans la classe Personne sont les suivantes ;

* using System.Runtime.Serialization; //Permet à la classe Personne d'être sérialisée.
* [Serializable()] //Indique que la classe est sérialisable.

## Conversions en double requérant une virgule comme délimiteur

### Description

Dans le formulaire frmPersonne.cs, lorsque l’on tentait d’entrer une valeur contenant des décimales séparées par un « . », on obtenait une erreur de conversion par la suite dans le programme.

En effet à ce moment clé du programme il faut absolument que le délimiteur pour les décimales soit une virgule, et non un point.

### Solution

Pour régler ce problème, il fait faire une recherche dans la textbox correpsondante, et y remplacer tous les points par des virgules.

Les lignes à rajouter dans la méthode button1\_Click de frmPersonne.cs ont les suivantes ;

char cA = ',';

char cB = '.';

tbxVolumePersonnel.Text = tbxVolumePersonnel.Text.Replace(cB, cA); //Permet de transformer les "." en "," pour que ça soit valide pour les conversions futures du double.

# Conclusions

Les objectifs du cahier des charges ont tous étés atteints. L’application est en effet capable de simuler une vente multiniveau graphiquement, nous avons même pris le temps d’ajouter des fonctionnalités en plus, en espérant qu’elles seront jugées en tant qu’amélioration de l’application.

J’ai eu du plaisir durant mon travail car j’apprécie la programmation en langage C#, c'est la chose qui m'a toujours intéressée le plus ici au Cifom, et j’espère en faire mon métier par la suite.

Un point important a été de comprendre comment fonctionnent les ventes multiniveau, ainsi que le composant TreeView.

Je suis maintenant plus à l’aise pour ce qui est de l’utilisation des ventes multiniveau, de tout ce qui touche aux TreeView, à la propriété tag que je ne conaissais pas.

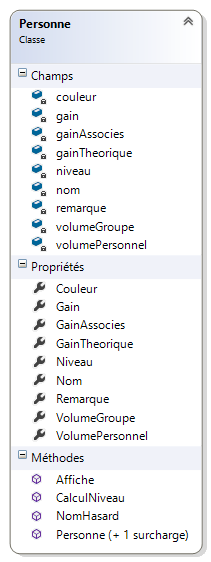
L’organisation quant aux tâches à effectuer pour chaque personne du groupe, et l’attribution d’un temps par tâche est important, c’est pourquoi nous avons fait attention à y réfléchir correctement dès le début. Cependant, il n’est pas si simple que ça de travailler efficacement en groupe, notamment lorsque quelqu’un fait une modification majeure, car il faut la répercuter immédiatement sur chacun des postes de travail, ce qui peut engendrer conflits et interruptions non désirées lors de réflexions diverses.

# Annexes

## Cahier des charges

Le cahier des charges est joint au document en tant qu’annexe.

## Diagrammes de classes



## Code source commenté

### frmPrincipal.cs

using System;

using System.Runtime;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.Serialization.Formatters.Binary; //Permet la Serialization

using System.IO;

using System.Drawing.Printing;

namespace ProjetArbre

{

public partial class frmPrincipal : Form

{

public frmPrincipal()

{

InitializeComponent();

}

Personne pMoi;

Personne P;

TreeNode tnCourant;

string strChemin = "";

private void Form1\_Load(object sender, EventArgs e)

{

pMoi = new Personne("Moi");

tnCourant = new TreeNode();

tnCourant.Tag = pMoi;

tnCourant.ToolTipText = pMoi.Affiche(true);

tnCourant.ContextMenuStrip = contextMenuStrip1;

tnCourant.ForeColor = pMoi.Couleur;

tvPrincipal.Nodes.Add(tnCourant);

Parcourir(tvPrincipal.Nodes[0]);

}

private void treeView1\_NodeMouseClick(object sender, TreeNodeMouseClickEventArgs e)

{

//Sélectionne la node cliquée

tvPrincipal.SelectedNode = e.Node;

//Intercepte le clic gauche de la souris

if (e.Button == MouseButtons.Right)

{

contextMenuStrip1.Show(tvPrincipal, e.Location);

}

}

private void ajouterPersonneToolStripMenuItem\_Click(object sender, EventArgs e)

{

P = new Personne();

tnCourant = new TreeNode();

tnCourant.Tag = P;

tnCourant.ToolTipText = P.Affiche(true);

tnCourant.ContextMenuStrip = contextMenuStrip1;

tnCourant.ForeColor = P.Couleur;

tvPrincipal.SelectedNode.Nodes.Add(tnCourant);

tvPrincipal.SelectedNode.Expand();

Parcourir(tvPrincipal.Nodes[0]); //On y envoie le début du Treeview

}

private void afficherDetailsStripMenuItem\_Click(object sender, EventArgs e)

{

P = (Personne)(tvPrincipal.SelectedNode.Tag);

MessageBox.Show(P.Affiche(false), "Affichage détails personne");

}

private void supprimerPersonneToolStripMenuItem\_Click(object sender, EventArgs e)

{

TreeNode PositionApres = tvPrincipal.SelectedNode.Parent; //Permet de se repositionner sur la bonne TreeNode après avoir supprimé la node à Supprimer.

if (tvPrincipal.SelectedNode.Level != 0) //Permet d'éviter la suppression du Noeud initial. A chaque sous-noeud, incrémente de 1. Donc empêche la supression du noeud père.

{

TreeNode tnClone = (TreeNode)tvPrincipal.SelectedNode.Clone();

for (int i = 0; i < tnClone.Nodes.Count; i++) //Parcours toutes les nodes du niveau en cours

{

tvPrincipal.SelectedNode.Parent.Nodes.Add(tnClone.Nodes[i]); //Ajoute toutes les notes du niveau en cours au Parent, ce qui va les mettre au meme niveau que les nodes du parent de celles supprimées.

}

tvPrincipal.SelectedNode.Remove(); //Finalement, supprime la node de base qui a servi de modèle à la copie.

tvPrincipal.SelectedNode = PositionApres;

}

tvPrincipal.ExpandAll();

Parcourir(tvPrincipal.Nodes[0]);

}

private void modifierPersonneToolStripMenuItem\_Click(object sender, EventArgs e)

{

P = (Personne)(tvPrincipal.SelectedNode.Tag); //Passage du TAG dans la personne

frmPersonne mod = new frmPersonne(P); //Création Form Dialogue

if (mod.ShowDialog() == DialogResult.OK)//Test si pression "OK" si oui attribuation de la personne modifier

{

P = mod.Pmod;

P.CalculNiveau();

tvPrincipal.SelectedNode.Tag = P;

tvPrincipal.SelectedNode.ToolTipText = P.Affiche(true);

tvPrincipal.SelectedNode.ForeColor = P.Couleur;

}

Parcourir(tvPrincipal.Nodes[0]);

}

private double Parcourir(TreeNode tnParent)

{

double dblTotal = 0;

double dblGainAss = 0; //Gain Associeé de l'enfant

//P = (Personne)(Parent.Tag); Ancienne ligne qui posait problème

Personne P = (Personne)(tnParent.Tag); //Création d'une autre personne car fonctionne mais le nom reste identique.

foreach (TreeNode tnEnfant in tnParent.Nodes)

{

dblTotal += Parcourir(tnEnfant);

Personne P2 = (Personne)(tnEnfant.Tag); //Création d'une autre personne car fonctionne mais le nom reste identique.

P2.CalculNiveau();

dblGainAss += P2.GainTheorique;

}

P.VolumeGroupe = dblTotal;

P.GainAssocies = dblGainAss;

dblTotal += P.VolumePersonnel;

P.CalculNiveau();

tnParent.Tag = P;

tnParent.Text = P.Nom + " " + (P.Gain).ToString("c");

tnParent.ForeColor = P.Couleur;

tnParent.ToolTipText = P.Affiche(true);

return dblTotal;

}

private void nouveauToolStripMenuItem\_Click(object sender, EventArgs e)

{

DialogResult drNouveau = MessageBox.Show("Voulez vous sauvegarder votre travail actuel ? \nTout travail non sauvegardé sera perdu!", "Nouveau projet", MessageBoxButtons.YesNoCancel);

if (drNouveau == DialogResult.Yes)

{

Sauvegarder();

}

else if (drNouveau == DialogResult.No)

{

//Restart le programme

tvPrincipal.Nodes[0].Remove();

tnCourant = new TreeNode();

tnCourant.Tag = pMoi;

tnCourant.ToolTipText = pMoi.Affiche(true);

tnCourant.ContextMenuStrip = contextMenuStrip1;

tnCourant.ForeColor = pMoi.Couleur;

tvPrincipal.Nodes.Add(tnCourant);

Parcourir(tvPrincipal.Nodes[0]);

}

}

private void enregistrerToolStripMenuItem\_Click(object sender, EventArgs e)

{

Sauvegarder();

}

private void ouvrirToolStripMenuItem\_Click(object sender, EventArgs e)//Problème : à l'ouverture le programme ne peut plus parcourir

{

DialogResult drOuvrir = MessageBox.Show("Voulez vous sauvegarder votre travail actuel ? \nTout travail non sauvegardé sera perdu!", "Ouvrir un projet", MessageBoxButtons.YesNoCancel);

if (drOuvrir == DialogResult.Yes)

{

Sauvegarder(); // Si la réponse de la personne est oui il va d'abbord y avoir la sauvegarde de l'arbre

Ouvrir();

}

else if (drOuvrir == DialogResult.No)

{

Ouvrir();

}

}

//Pour l’aide je suis partie de se site : http://stackoverflow.com/questions/5868790/saving-content-of-a-treeview-to-a-file-and-load-it-later

private void Sauvegarder()

{

SaveFileDialog sfdEnregistrer = new SaveFileDialog();

sfdEnregistrer.DefaultExt = "txt";

sfdEnregistrer.Filter = "Fichier Text (\*.txt)|\*.txt"; //Permet de filtrer l'affichage

sfdEnregistrer.FileName = strChemin;

sfdEnregistrer.ShowDialog();

if (sfdEnregistrer.FileName.Length > 0)

{

using (Stream sFile = File.Open(sfdEnregistrer.FileName, FileMode.Create))

{

BinaryFormatter bf = new BinaryFormatter();

bf.Serialize(sFile, tvPrincipal.Nodes.Cast<TreeNode>().ToList());

}

strChemin = sfdEnregistrer.FileName;

}

}

private void Ouvrir()

{

try

{

OpenFileDialog ofdOuvrir = new OpenFileDialog();

ofdOuvrir.DefaultExt = "txt";

ofdOuvrir.Filter = "Fichier Text (\*.txt)|\*.txt"; //Permet de filtrer l'affichage

ofdOuvrir.ShowDialog();

if (ofdOuvrir.FileName.Length > 0)

{

using (Stream sFile = File.Open(ofdOuvrir.FileName, FileMode.Open))

{

tvPrincipal.SelectedNode = tvPrincipal.Nodes[0];

tvPrincipal.SelectedNode.Remove();

BinaryFormatter bf = new BinaryFormatter();

object obj = bf.Deserialize(sFile);

TreeNode[] nodeList = (obj as IEnumerable<TreeNode>).ToArray();

tvPrincipal.Nodes.AddRange(nodeList);

strChemin = ofdOuvrir.FileName;

}

tvPrincipal.ExpandAll();

}

}

catch

{

MessageBox.Show("Problème lors du chargement du fichier, veuillez réessayer", "Erreur");

Application.Restart();

Ouvrir();

}

}

private void treeView1\_KeyDown(object sender, KeyEventArgs e)

{

//Va capturer les touches pressées, afin de réagir aux raccourcis claviers courants.

//La méthode PerformClick() simule un clic sur un bouton

if(e.KeyData == (Keys.Control | Keys.N))

{

tsmNouveau.PerformClick();

}

if (e.KeyData == (Keys.Control | Keys.O))

{

tsmOuvrir.PerformClick();

}

if (e.KeyData == (Keys.Control | Keys.S))

{

if (strChemin != "")

{

tsmEnregistrerRapide.PerformClick();

}

else

{

Sauvegarder();

}

}

}

private void enregisterRapideToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (strChemin != "")

{

using (Stream file = File.Open(strChemin, FileMode.Create))

{

BinaryFormatter bf = new BinaryFormatter();

bf.Serialize(file, tvPrincipal.Nodes.Cast<TreeNode>().ToList());

}

}

else

{

Sauvegarder();

}

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

DialogResult drFermeture = MessageBox.Show("Voulez vous sauvegarder votre travail actuel ? \nTout travail non sauvegardé sera perdu!", "Fermeture du programme", MessageBoxButtons.YesNoCancel);

if (drFermeture == DialogResult.Yes)

{

Sauvegarder(); //Sauvegarde avant fermeture du programme

}

else if (drFermeture == DialogResult.No)

{

//Fermeture du programme

}

else if (drFermeture==DialogResult.Cancel)

{

e.Cancel = true;//Annulation de la fermeture

}

}

}

}

### frmPersonne.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ProjetArbre

{

public partial class frmPersonne : Form

{

public Personne Pmod;

bool bValideVol = true;

bool bValideNom = true;

public frmPersonne(Personne P)

{

InitializeComponent();

AcceptButton = btnOk; //A la pression du bouton "Enter", va simuler un clic sur la touche Ok.

Pmod = P;

tbxNom.Text = P.Nom;

tbxVolumePersonnel.Text = Convert.ToString(P.VolumePersonnel);

lblVolumeGroupeValue.Text = P.VolumeGroupe.ToString("c");

double dblNiveau = P.Niveau / 100; //Variable utilisée suite a l'erreur d'affichage sur le formulaire de modification Car affichais 300,00% au lieux de 3,00%

lblNiveauActuelValue.Text = dblNiveau.ToString("p");

lblGainValue.Text = P.Gain.ToString("c");

rtbxRemarque.Text = P.Remarque;

}

private void button1\_Click(object sender, EventArgs e)

{

char cA = ',';

char cB = '.';

tbxVolumePersonnel.Text = tbxVolumePersonnel.Text.Replace(cB, cA); //Permet de transformer les "." en "," pour que ça soit valide pour les conversions futures du double.

double dblVol = Convert.ToDouble(tbxVolumePersonnel.Text);

dblVol = ((int)((dblVol \* 20) + 0.5));//Arrondis de la monnaie au 5ct près

dblVol /= 20;

Pmod.Remarque = rtbxRemarque.Text;

Pmod.Nom = tbxNom.Text;

Pmod.VolumePersonnel = dblVol;

}

private void tbxVolumePersonnel\_Leave(object sender, EventArgs e)

{

if (Regex.IsMatch(tbxVolumePersonnel.Text, @"^[0-9]\*(?:[\, , \.][0-9]?[0-9])?$") && tbxVolumePersonnel.Text!="")

{

bValideVol = true;

tbxVolumePersonnel.ForeColor = Color.Black;

if (bValideNom == true && bValideVol == true)

{

btnOk.Enabled = true;

}

else

{

btnOk.Enabled = false;

}

}

else

{

bValideVol = false;

tbxVolumePersonnel.ForeColor = Color.Red;

btnOk.Enabled = false;

}

}

private void tbxNom\_TextChanged(object sender, EventArgs e)

{

if (Regex.IsMatch(tbxNom.Text, @"^[A-Z,a-z, ,\-,.]\*?$") && tbxNom.Text != "")

{

bValideNom = true;

tbxNom.ForeColor = Color.Black;

if (bValideNom == true && bValideVol == true)

{

btnOk.Enabled = true;

}

else

{

btnOk.Enabled = false;

}

}

else

{

bValideNom = false;

tbxNom.ForeColor = Color.Red;

btnOk.Enabled = false;

}

}

}

}

### Personne.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading;

using System.Drawing;

using System.Runtime.Serialization; //Permet à la classe d'être Sérialisée

namespace ProjetArbre

{

[Serializable()] //Permet à la classe d'être Sérialisée

public class Personne

{

private Color couleur;

private double gain;//Ce qu'on a gangé au final

private double gainAssocies; //Gain des enfants

private double gainTheorique; //Ce qu'on aura gangé si la distribution n'aurait pas été faite

private double niveau;

private string nom;

private string remarque;

private double volumeGroupe;

private double volumePersonnel;

public Personne()

{

NomHasard();

volumePersonnel = 500.00; //Valeur fixe pour tous.

}

public Personne(string strNom)

{

nom = strNom;

volumePersonnel = 500.00;

}

public string Affiche(bool bTooltip)

{

if (bTooltip == true) //Si c'est pour le tooltiptext, la remarque ne doit pas être présente.

{

return nom + "\nNiveau: " + niveau + "\nPerso: " + volumePersonnel.ToString("c") + "\nGroupe: " + volumeGroupe.ToString("c") + "\nGain Théorique: " + gainTheorique.ToString("c") + "\nGain net: " + gain.ToString("c") + "\nGain associés: " + gainAssocies.ToString("c");

}

else

{

return nom + "\nNiveau:\t\t" + niveau + "\nPerso:\t\t" + volumePersonnel.ToString("c") + "\nGroupe:\t\t" + volumeGroupe.ToString("c") + "\nGain Théorique:\t" + gainTheorique.ToString("c") + "\nGain net:\t\t" + gain.ToString("c") + "\nGain associés:\t" + gainAssocies.ToString("c") + "\nRemarque:\n" + remarque;

}

}

public void CalculNiveau()

{

double dblChiffre = VolumeGroupe + VolumePersonnel;

//Si aucune de ces catégories, le niveau retourne à zero.

if (dblChiffre < 500)

{

Niveau = 1;

Couleur = Color.Black;

}

else if (dblChiffre >= 500 && dblChiffre < 1500)

{

Niveau = 3;

Couleur = Color.DarkGoldenrod;

}

else if (dblChiffre >= 1500 && dblChiffre < 2500)

{

Niveau = 6;

Couleur = Color.Cyan;

}

else if (dblChiffre >= 2500 && dblChiffre < 4800)

{

Niveau = 9;

Couleur = Color.Firebrick; //Ancien yellow mais la couleur ne se lisait pas bien

}

else if (dblChiffre >= 4800 && dblChiffre < 6500)

{

Niveau = 12;

Couleur = Color.Maroon;

}

else if (dblChiffre >= 6500 && dblChiffre < 10000)

{

Niveau = 15;

Couleur = Color.Salmon;

}

else if (dblChiffre >= 10000 && dblChiffre < 20000)

{

Niveau = 18;

Couleur = Color.Green;

}

else

{

Niveau = 20;

Couleur = Color.Red;

}

GainTheorique = dblChiffre \* Niveau /100;

Gain = GainTheorique - GainAssocies;

}

public void NomHasard()

{

int iNum;

char cLettre;

Random \_random = new Random();//créadtion du random

for (int iCpt = 0; iCpt <= 3; iCpt++) //Genere un niom de 4 lettres

{

if (iCpt == 1 || iCpt == 3) //Permet de ne generer que des voyelles pour les lettres numero 2 et 4

{

do

{

iNum = \_random.Next(0, 26); // Zero to 25

} while (iNum != 0 && iNum != 4 && iNum != 8 && iNum != 14 && iNum != 20 && iNum != 24);//test pour savoir voyelle

}

else

iNum = \_random.Next(0, 26); // Zero to 25

if(iCpt==0)

cLettre = (char)('A' + iNum);//transorme le random en lettre majuscule pour le premier caractère

else

cLettre = (char)('a' + iNum);//transorme le random en lettre

nom += Convert.ToString(cLettre);//insère la lettre au nom

}

}

public Color Couleur

{

get { return couleur; }

set { couleur = value; }

}

public double Gain

{

get { return gain; }

set { gain = value; }

}

public double GainAssocies

{

get { return gainAssocies; }

set { gainAssocies = value; }

}

public double GainTheorique

{

get { return gainTheorique; }

set { gainTheorique = value; }

}

public double Niveau

{

get { return niveau; }

set { niveau = value; }

}

public string Nom

{

get { return nom; }

set { nom = value; }

}

public string Remarque

{

get { return remarque; }

set { remarque = value; }

}

public double VolumeGroupe

{

get { return volumeGroupe; }

set { volumeGroupe = value; }

}

public double VolumePersonnel

{

get { return volumePersonnel; }

set { volumePersonnel = value; }

}

}

}

## Références

Sauvegarde / Ouverture projet :<http://stackoverflow.com/questions/5868790/saving-content-of-a-treeview-to-a-file-and-load-it-later>

Fonction récursive :  
<http://www.dotnetperls.com/recursion>

Apprentissage des Expression régulière :  
<http://msdn.microsoft.com/fr-fr/library/hs600312(v=vs.110).aspx>

Annuler la fermeture d’un formulaire :  
<http://msdn.microsoft.com/fr-fr/library/system.windows.forms.form.formclosing(v=vs.110).aspx>

Prise de connaissance sur le treeview :  
<http://msdn.microsoft.com/fr-fr/library/system.windows.forms.treeview(v=vs.110).aspx>

Remplace un caractère dans une string :  
<http://msdn.microsoft.com/fr-fr/library/fk49wtc1(v=vs.110).aspx>