



DOSSIER DE PROGRAMMATION N°3

Communautés

RESUME

Amélioration du DP2 afin d'intégrer la gestion de plusieurs joueurs et d'implémenter une nouvelle fonctionnalité : les communautés.

Saint Adjutor, Vernon

TABLE DES MATIERES

1	Présentation du dossier.....	2
1.1	Présentation globale	2
1.2	Liste de compétences.....	2
2	Structures de données	3
2.1	Structure des form	3
2.2	Structure des classes	4
3	Explication du code	6
3.1	Algorithmes redondants.....	6
3.2	Permettre aux joueurs de se créer une identité en ligne.....	8
3.2.1	Changement apportés au précédent dossier de programmation.....	8
3.2.2	Créer une communauté	11
3.2.3	Rejoindre une communauté.....	12
3.2.4	Quitter sa communauté	13
3.2.5	Supprimer une communauté	13
3.2.6	Voir une communauté.....	14
3.2.7	Modifier la communauté.....	15
3.3	Fournir un classement des membres de la communauté	16
3.4	Faciliter les échanges et le regroupement	17
3.5	Modérer le forum	18
3.6	Animer la communauté.....	19
4	Jeu d'essai.....	20
4.1	Les différents essais effectués.....	20
6	Annexes	23
6.1	Code : Créer une communauté	23
6.2	Code : Quitter sa communauté	24
6.3	Code : Modifier la communauté	25
6.4	Code : Classement	26
6.5	Code : Chat	26
6.6	Code : Influenceurs.....	27



1 PRESENTATION DU DOSSIER

1.1 PRESENTATION GLOBALE

La société BABYLONE pour laquelle nous avons réalisé le dossier de programmation numéro 2, contente de notre travail, nous a recontactés afin de produire une évolution de ce même travail. Ce travail avait dans un premier temps pour but de créer une interface de type MDI (Multiple Document Interface) permettant de gérer des joueurs (mais nous n'avions jusqu'alors qu'à réaliser un seul joueur appelé JoueurTest dans ce projet). L'utilisateur devait pouvoir via cette interface graphique :

- Créer un joueur (Objet de la classe Joueur possédant : nom, prénom, mail, pseudonyme et avatar)
- Voir le joueur créé
- Modifier son adresse mail
- Modifier son avatar
- Faire changer son niveau
- Faire changer son score

La nouvelle demande qui nous a été adressée reprend les codes de l'ancienne mais ceux-ci nécessitent tout de même quelques modifications. De nouvelles demandes ont été demandées il y aura donc également de nouvelles structures de données (classes et form) afin de répondre aux besoins de la société. La société demande les améliorations suivantes :

- Permettre aux joueurs de se créer une identité en ligne
- Fournir un classement des membres de la communauté
- Faciliter les échanges et le regroupement
- Modérer le forum de discussion
- Animer la communauté

1.2 LISTE DE COMPETENCES

Les compétences associées sont les suivantes :

- A4.1.6 Gestion d'environnements de développement et de test
- A4.1.7 Développement, utilisation et adaptation de composants logiciels
- A4.1.8 Réalisation des tests nécessaires à la validation d'éléments adaptés ou développés
- A4.1.9 Rédaction d'une documentation technique



2 STRUCTURES DE DONNEES

Le langage utilisé dans ce projet est le C# en projet Windows Form. Le projet se compose de 5 classes et de 16 form, 5 de celles-ci sont reprise du dossier de programmation numéro 2 mais celles-ci ont subies des modifications.

2.1 STRUCTURE DES FORM

Une form appelée **frmMenu** est comme son nom l'indique la form de démarrage et contient le menu MDI qui lui aussi a été modifié pour répondre aux exigences du projet imposées par la société. Ce menu possède une barre de navigation positionnée sur le bord supérieur de la form et permettant d'accéder aux autres form. Elle se compose de 3 sous-menus dont un possédant lui-même 2 sous-menus. Ces menus sont intitulés : *Jeu*, *Joueur* et *Communauté*.



Exemple de form :



MESLIN Thibaut
BTS SIO
Saint Adjutor VERNON

2.2 STRUCTURE DES CLASSES

La classe sur laquelle le projet repose est la classe **Globale**. Elle contient toutes les informations utiles au bon fonctionnement du projet comme le nombre maximum de points d'un niveau ou encore les différentes instanciations et collections utiles.

```
99+ références
class Globale
{
    public static int nbMaxNiveaux = 5;           //Nombre maximum de niveaux
    public static int plafondScore = 10;         //Score maximum par niveau
    public static List<Joueur> lesJoueurs = new List<Joueur>(); //Collection contenant les joueurs
    public static List<Communaute> lesCommunautes = new List<Communaute>(); //Collection contenant les communautes
    public static List<Message> lesMessages = new List<Message>(); //Collection contenant les messages de chat
}
```

La classe **Joueur** contient toutes les informations relatives aux joueurs tels que leurs nom, prénom ou encore le nombre de messages postés ou insultes tentées (cf. **frmChat**). Elle contient également le constructeur de cette classe ainsi que tous les accesseurs et mutateurs relatifs aux informations de l'utilisateur. Elle contient également des procédures relatives aux scores et niveaux mais aussi au chat de communauté.

```
27 références
class Joueur
{
    private string nom;           //Nom du joueur
    private string prenom;        //Prenom du joueur
    private string mail;          //Mail du joueur
    private string pseudo;        //Pseudo du joueur
    private Image avatar;         //Avatar du joueur
    private int niveau;           //Niveau du joueur
    private int score;            //Score du joueur
    private string communaute;    //Communauté à laquelle appartient le joueur (si aucune "")
    private int nbMessage;        //Nombre de messages postés
    private int nbInjureEssaye;   //Nombre d'injures voulant êtres postées
}
```

La classe **Communauté** contient toutes les informations relatives aux communautés de joueurs. Parmi ces informations on retrouve par exemple le nom et le logo de la communauté. De même on retrouve le constructeur et tous les accesseurs et mutateurs relatifs aux informations des communautés. Elle contient également les méthodes d'ajout et de retrait d'un joueur à la communauté.

```
11 références
class Communaute
{
    private string nom;           //Nom de la communaute
    private string fondateur;     //Pseudo du fondateur
    private Image logo;           //Logo de la communaute
    private DateTime dateCreation; //Date de création de la communaute
    private List<Joueur> lesMembres; //Collections contenant les membres de la communaute
}
```



La classe **Message** contient toutes les informations relatives aux messages comme la personne qui l'a envoyé ou encore la date et l'heure de l'envoi. Les accesseurs et mutateurs pour ces informations ainsi que le constructeur y sont également présents.

```
4 références
class Message
{
    private DateTime datePublication; //Date de publication du message
    private string joueur;           //Joueur qui a publié
    private string contenu;           //Contenu du message
}
```

La classe **Comparer** contient seulement des procédures utiles lors du classement des joueurs dans la communauté. Elles servent à ordonner les joueurs de la communauté selon leur niveau d'avancement dans le jeu. Elle contient une classe nommée **comparerJoueurNiveauDecroissant** qui permet en fonction du nombre retourné de savoir si le joueur est moins, plus ou pareillement avancé dans le jeu. Elle est alors reprise dans la méthode de classe **TrierNiveau()** utilisant un **Icomparer<...>** de joueurs afin de modifier la position des objets de la classe Joueur présent dans la collection des joueurs.

```
1 référence
class Comparer
{
    1 référence
    public class comparerJoueurNiveauDecroissant : Comparer<Joueur> //Permet de retourner un nombre en fonction du niveau du joueur
    {
        0 références
        public override int Compare(Joueur x, Joueur y)
        {
            if (x.getNiveau() == y.getNiveau()) return 0;
            return (x.getNiveau() < y.getNiveau()) ? 1 : -1;
        }
    }

    1 référence
    public static void TrierNiveau() //Permet de trier les joueurs par niveau et ordre décroissant
    {
        IComparer<Joueur> critereTri = new comparerJoueurNiveauDecroissant();
        Globale.lesJoueurs.Sort(critereTri);
    }
}
```



3 EXPLICATION DU CODE

Par soucis de simplification, les explications ne seront pas faites actions demandés par actions demandés mais form par action ou form, l'explication d'un algorithme s'étendant parfois sur plusieurs form du programme je pense que cela sera plus compréhensible ainsi.

3.1 ALGORITHMES REDONDANTS

Ce programme comporte quelques codes redondants présents dans la grande majorité des form. Ce code est assez simple mais est également très utile pour éviter que le programme plante sans raison apparente.

Le code concerne le déblocage du bouton servant à valider l'action désirée (appelé btValider et présentant comme inscription "VALIDER" dans la quasi-totalité du code). Ce code est sensiblement différent d'une form à une autre parce que ce ne sont pas toujours les mêmes informations qui sont demandés ou sujettes à modifications. Les seules choses qui changent sont donc les conditions dans la boucle **if(...)** / **else** présente dans cette méthode. Si les conditions indiquées sont remplies alors le bouton se débloquent sinon il restera bloqué ou se re-bloquera. Ce code est appelé **DeblocageBoutonValider()** dans toutes les form l'utilisant et est déclaré directement après l'initialisation des composant de la form. Elle est appelée dans les évènements **textChanged** ou **indexChanged** des éléments graphiques.

Exemple du code pour la form **frmNvCommunaute** :

```
3 références
public void DeblocageBoutonValider()
{
    //Si toutes les informations sont remplies alors
    if (tbNom.Text != "" && pbLogo.Image != null && cbJoueur.Text != "")
    {
        btValider.Enabled = true;    //On débloque le bouton
    }
    else
    {
        btValider.Enabled = false;
    }
}
```



Le second code redondant est le code d'ouverture des form via le menu MDI, ici **frmMenu**. Ce code est présent dans cette form uniquement. Il définit la façon dont on ouvre la form mais aussi déclare que **frmMenu** est la form PARENTE de celle qui va s'ouvrir et enfin de l'ouvrir. Il s'effectue lors de l'appui sur un bouton du menu MDI.

Exemple pour l'ouverture de **frmNvJoueur** :

```
frmNvJoueur maForm = new frmNvJoueur();
maForm.MdiParent = this;
maForm.Show();
```

Le troisième code est celui de fermeture de la form actuel. Il s'effectue lors de l'appui sur le bouton "QUITTER".

Exemple pour la form **frmNvCommunaute** :

```
this.Close(); //On ferme frmNvCommunaute
```

Le quatrième code est utilisé lorsqu'une action externe au programme doit être effectuée, dans notre code ces actions sont la sélection d'une image afin qu'elle devienne avatar de joueur ou logo de communauté. Ce code est exécuté lors de l'appui sur un bouton comme "Logo" ou "Avatar". Il s'ouvre alors sur l'ordinateur de l'utilisateur une page d'explorateur de fichier permettant à celui-ci de sélectionner une image dans ses fichiers afin de l'ouvrir dans le programme. Un filtre est appliqué pour que seuls les extensions d'images tels que jpeg ou png puissent être ouvertes.

```
OpenFileDialog dialog = new OpenFileDialog(); //On instancie un nouvel objet de la classe OpenFileDialog
dialog.Filter = "Image files(*.jpg, *.jpeg, *.jpe, *.jfif, *.png) | *.jpg; *.jpeg; *.jpe; *.jfif; *.png"; //On filtre pour avoir uniquement les images
if (dialog.ShowDialog() == DialogResult.OK) //Si quelque chose à été ouvert alors
{
    Image avatar = Image.FromFile(dialog.FileName); //On instancie un objet de la classe Image contenant l'image ouverte depuis l'ordinateur
    pbLogo.Image = avatar; //On l'affiche dans la PictureBox
}
```

Le dernier code est celui du bouton "VALIDER". Celui-ci est le plus redondant car présent à chaque form. De même que pour la méthode de déblocage plus haut seules les informations modifiées ou ajoutées ne sont différentes. Dans la globalité ces algorithmes font appels aux mutateurs des objets concernés par la ou les modifications. Tous les boutons font appels à la classe **Globale** et des informations contenues dans celle-ci. S'il existe des exceptions dans ces codes ils seront précisés dans la description de celui-ci.

Exemple pour la form **frmModifAvatar** :

```
int position = cbJoueur.SelectedIndex;
Globale.lesJoueurs.ElementAt(position).setAvatar(pbNvAvatar.Image); //On stocke l'avatar à la place de l'ancien dans la collection lesJoueurs
pbActAvatar.Image = Globale.lesJoueurs.ElementAt(position).getAvatar(); //On affiche l'avatar dans la textBox avatar actuel
pbNvAvatar.Image = null; //On vide la PictureBox nouvel avatar
```



3.2 PERMETTRE AUX JOUEURS DE SE CREER UNE IDENTITE EN LIGNE

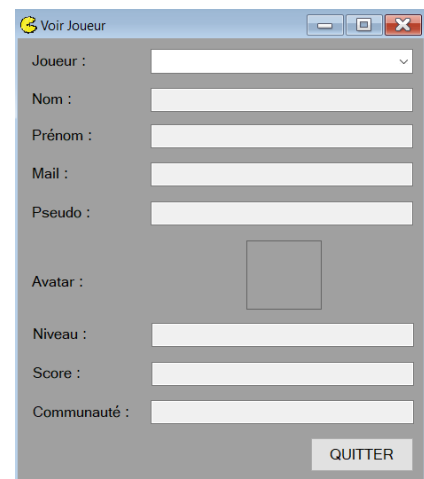
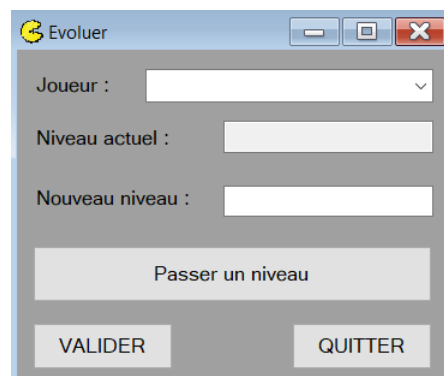
3.2.1 Changement apportés au précédent dossier de programmation

Le projet précédent était codé de sorte que seul un joueur soit utilisé à l'inverse de celui-ci où on veut pouvoir en gérer autant que rentrés. Il a donc fallu trouver un moyen afin de pouvoir tous les gérer, ce moyen a été l'utilisation de collections permettant de regrouper plusieurs éléments dans un groupe. Ces éléments une fois rentrés dans la collection se voient attribué un index permettant de les repérer et de pouvoir les atteindre afin de récupérer les informations sur le joueur à l'index choisi.

Afin de simplifier l'utilisation de celles-ci et de fluidifier la façon d'afficher et de rechercher les informations, l'implémentation de comboBox dans les form s'imposaient. Ces éléments graphiques regroupent dans une liste déroulante toutes les informations qui y sont entrés. On peut donc des lors charger le contenu des collections dans les différents comboBox présents sur les form. Il sera fait de même pour les communautés plus tard.

La modification du dossier de programmation précédent était donc alors obligatoire afin de rendre simple et fluide l'utilisation des interfaces et de l'application. Les 6 form présentes dans le précédent projet ont donc été mises à jour. Les form modifiées sont les suivantes : **frmNvJoueur**, **frmVoirJoueur**, **frmModifMail**, **frmModifAvatar**, **frmEvoluer** et **frmPoints**. Ces 6 form ont été regroupées dans 2 sous-menus appelés *Joueur* et *Jeu* dans **frmMenu** (cf. [2.1 Structure des form](#)).

Exemple de changements :



Les codes ont donc inévitablement été modifiés et/ou agrémenté de nouvelles lignes. Parmi celles-ci on retrouve le code d'ajout à la collection *lesJoueurs* de la classe **Globale** à l'aide d'un **.Add(...)**. On retrouve ce changement dans la form **frmNvJoueur**.

```
//On instance leJoueurTest présent dans la classe Globale avec le constructeur de la classe Joueur
Globale.lesJoueurs.Add(new Joueur(tbNom.Text, tbPrenom.Text, tbMail.Text, tbPseudo.Text, pbAvatar.Image));
//On informe l'utilisateur que le joueur à bien été créé
MessageBox.Show("Le joueur à bien été créé", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

On retrouve également dans toutes les autres form le code de chargement de la comboBox. Via ce code on peut charger les Joueurs présents dans *lesJoueurs* de la classe **Globale** en utilisant un **foreach(...)**.

```
//On affiche tous les joueurs qui ont été créés
foreach (Joueur unJoueur in Globale.lesJoueurs)
{
    cbJoueur.Items.Add(unJoueur.getPseudo() + " (" + unJoueur.getNom() + " " + unJoueur.getPrenom() + ")");
}
```

On retrouve également tout ce qui servait avant à modifier les informations de l'objet *JoueurTest* qui a été remplacé par la collection *lesJoueurs*. On retrouve donc tous les mutateurs et accesseurs de cet ancien objet, on va donc maintenant passer par la classe **Globale** afin d'atteindre la collection *lesJoueurs* qu'elle stocke et donc pouvoir accéder à la classe **Joueur** car celle-ci est une collection d'objet de cette classe. Ces mutateurs et accesseurs interviennent généralement dans les codes de bouton "VALIDER" ou dans les comboBox, dans les 2 cas on retrouve le même procédé de recherche qui se compose d'une boucle **While + booléen** permettant de rechercher l'élément à modifier ou à afficher (essentiellement utilisé pour l'affichage).

Exemples de changement de mutateurs ou accesseurs :

```
int position = cbJoueur.SelectedIndex;
if (tbNvMail.Text.Contains("@") == true && tbNvMail.Text.Contains(".") == true) //Si le nouveau mail contient un "." et un "@" alors il est valide donc
{
    Globale.lesJoueurs.ElementAt(position).setMail(tbNvMail.Text); //On stocke le nouveau mail à la place de l'ancien
    tbActMail.Text = Globale.lesJoueurs.ElementAt(position).getMail(); //On affiche le nouveau mail dans la textBox du mail actuel
    tbNvMail.Clear(); //On vide la textBox nouveau mail
}
else
{
    //Sinon on prévient l'utilisateur que son adresse email n'est pas valide
    MessageBox.Show("Le mail ne répond pas aux critères de validités, veuillez vérifier", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

```
int position = cbJoueur.SelectedIndex;
Globale.lesJoueurs.ElementAt(position).setAvatar(pbNvAvatar.Image); //On stocke l'avatar à la place de l'ancien dans la collection lesJoueurs
pbActAvatar.Image = Globale.lesJoueurs.ElementAt(position).getAvatar(); //On affiche l'avatar dans la textBox avatar actuel
pbNvAvatar.Image = null; //On vide la PictureBox nouvel avatar
```



Exemples de boucle **While** + **booléen** :

```
int position = cbJoueur.SelectedIndex; //On recupère l'index de l'élément choisi
int idx = 0; //Variable utile au parcours
bool trouve = false; //Variable utile à l'affichage

//Tant que la collection n'as pas été parcourue totalement et que trouve est faux alors
while (idx < Globale.lesJoueurs.Count && !trouve)
{
    //Si l'index de l'élément choisi est égal à la variable de parcours alors
    if (position == idx)
    {
        trouve = true; //Trouve passe à true
    }
    else
    {
        idx++; //On regarde l'élément suivant
    }
}

//Si on à trouvé quelque chose alors on affiche
if (trouve)
{
    pbActAvatar.Image = Globale.lesJoueurs.ElementAt(position).getAvatar();
}
//Sinon on informe l'utilisateur
else
{
    MessageBox.Show("Un problème est survenu avec ce joueur, veuillez réessayer", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

```
int position = cbJoueur.SelectedIndex; //On recupère l'index de l'élément choisi
int idx = 0; //Variable utile au parcours
bool trouve = false; //Variable utile à l'affichage

//Tant que la collection n'as pas été parcourue totalement et que trouve est faux alors
while (idx < Globale.lesJoueurs.Count && !trouve)
{
    //Si l'index de l'élément choisi est égal à la variable de parcours alors
    if (position == idx)
    {
        trouve = true; //Trouve passe à true
    }
    else
    {
        idx++; //On regarde l'élément suivant
    }
}

//Si on à trouvé quelque chose alors on affiche
if (trouve)
{
    tbNom.Text = Globale.lesJoueurs.ElementAt(position).getNom();
    tbPrenom.Text = Globale.lesJoueurs.ElementAt(position).getPrenom();
    tbMail.Text = Globale.lesJoueurs.ElementAt(position).getMail();
    tbPseudo.Text = Globale.lesJoueurs.ElementAt(position).getPseudo();
    pbAvatar.Image = Globale.lesJoueurs.ElementAt(position).getAvatar();
    tbNiveau.Text = Globale.lesJoueurs.ElementAt(position).getNiveau().ToString();
    tbScore.Text = Globale.lesJoueurs.ElementAt(position).getScore().ToString();
    tbCommunaute.Text = Globale.lesJoueurs.ElementAt(position).getCommunaute();

    //Si le joueur n'appartient pas à une communaute
    if (tbCommunaute.Text == "")
    {
        tbCommunaute.Text = "AUCUNE"; //On affiche aucune
    }
}
//Sinon on informe l'utilisateur
else
{
    MessageBox.Show("Un problème est survenu avec ce joueur, veuillez réessayer", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```



3.2.2 Créer une communauté

Avant de pouvoir rejoindre une communauté, il faut d'abord que celle-ci existe et on doit donc retrouver une form permettant de créer une communauté, ici **frmNvCommunaute**. Cette action est disponible dans le sous-menu *Communaute / Gestion* du menu MDI par l'ouverture de la form **frmNvCommunaute**. Cette form nous permet en rentrant les informations nécessaires de créer une communauté. Le constructeur de la classe **Communaute** impose que soient renseignés : nom, fondateur et logo de celle-ci.



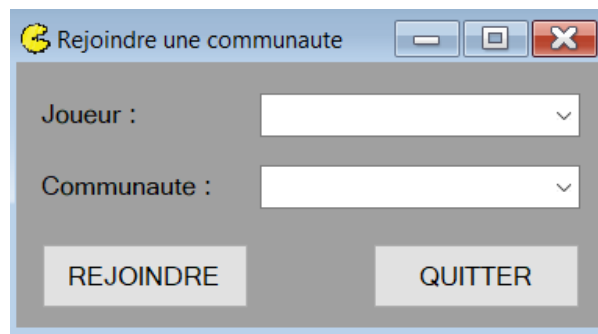
Nous partirons du principe qu'un joueur ne peut appartenir qu'à une seule communauté et qu'il ne peut donc pas en créer une en étant lui-même membre d'une. On part également du principe que une communauté à un nom unique et qu'il ne peut donc pas se trouver 2 communautés portant le même nom, cela sera évité par une recherche en **While + booléen** (cf. [3.2 Algorithme redondant](#)). Lors de l'appui sur le bouton "VALIDER", un objet de la classe **Communaute** sera instancié et prendra les informations rentrées ou sélectionnées par l'utilisateur. Il sera ensuite ajouté à la collection *lesCommunautes* de la classe **Globale** afin de pouvoir y accéder n'importe quand. On estime que le fondateur de la communauté rejoint sa communauté, on l'ajoute donc à la liste des membres.

Pour voir le code : cf. [6.1 Code : Créer une communauté](#).



3.2.3 Rejoindre une communauté

Lorsque l'on est un joueur désireux de jouer avec les autres ou de voir d'autres joueurs mais qu'il ne souhaite pas créer une communauté, il peut simplement rejoindre une qui a déjà été créée auparavant par quelqu'un d'autre. Cette action est disponible dans le sous-menu *Joueur* du menu MDI par l'ouverture de la form **frmRejoindreCommunaute**. Cette form nous permet de sélectionner un joueur et une communauté via 2 comboBox par l'appui du bouton valider celui-ci rejoindra la communauté à moins qu'il ne soit lui-même dans une communauté dans ce cas il devra la quitter d'abord. Comme il a rejoint la communauté, il met en lui le nom de sa communauté grâce au mutateur **setCommunaute(...)** et on l'ajoute à la liste des membres de sa communauté.



```
int position = cbJoueur.SelectedIndex; //On recupère l'index de l'élément choisi
int idx = 0; //Variable utile au parcours
bool trouve = false; //Variable utile à l'affichage

//Tant que la collection n'as pas été parcourue totalement et que trouve est faux alors
while (idx < Globale.lesJoueurs.Count && !trouve)
{
    //Si l'index de l'élément choisi est égal à la variable de parcours alors
    if (position == idx)
    {
        trouve = true; //Trouve passe à true
    }
    else
    {
        idx++; //On regarde l'élément suivant
    }
}

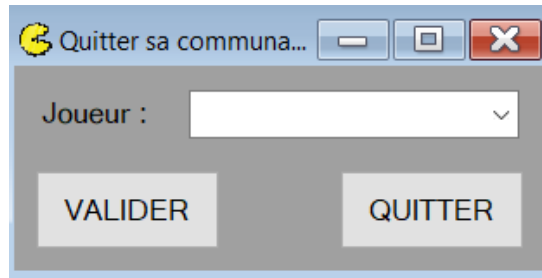
//Si on à trouvé le joueur alors
if (trouve)
{
    //Si le joueur n'a pas de communauté alors
    if (Globale.lesJoueurs.ElementAt(position).getCommunaute() != "")
    {
        //On informe l'utilisateur
        MessageBox.Show("Ce joueur appartient déjà une communauté", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        //On renseigne le nom de la communauté dans l'objet
        Globale.lesJoueurs.ElementAt(position).setCommunaute(Globale.lesCommunautes.ElementAt(cbCommunaute.SelectedIndex).getNom());
        //On l'ajoute à la liste des membres
        Globale.lesCommunautes.ElementAt(cbCommunaute.SelectedIndex).ajouterMembre(Globale.lesJoueurs.ElementAt(position));
        //On informe l'utilisateur
        MessageBox.Show("Vous avez rejoint la communauté", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

//Sinon on informe l'utilisateur
else
{
    MessageBox.Show("Un problème est survenu avec ce joueur, veuillez réessayer", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```



3.2.4 Quitter sa communauté

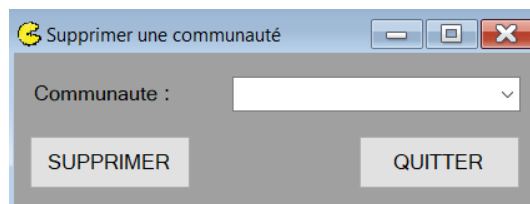
Lorsque l'on veut changer de communauté ou pouvoir créer sa propre communauté et que l'on appartient déjà à l'une d'entre elles, il est nécessaire de la quitter. Cette action est disponible via le sous-menu *Joueur* du menu MDI par l'ouverture de la form **frmQuitterCommunaute**. La form se compose seulement d'un comboBox afin de sélectionner le joueur qui est désireux de quitter sa communauté. Lors de l'appui sur le bouton "VALIDER", on va chercher si le joueur possède une communauté, s'il n'en possède pas il n'a rien à quitter on l'en informe donc sinon on remet sa communauté à vide. On estime que si la communauté est vide, elle ne sert à rien donc on la supprime.



Pour voir le code : cf. [6.2 Code : Quitter sa communau](#)

3.2.5 Supprimer une communauté

Lorsque l'on veut supprimer une communauté, il faut réassigner toutes les informations de l'attribut privé communauté à vide pour signifier que le joueur n'a plus de communauté. Cette action est disponible via le sous-menu *Communauté / Gestion* du menu MDI par l'ouverture de la form **frmSupCommunaute**. On doit également retirer la communauté de la collection *lesCommunautes* de la classe **Globale**.



```
int position = cbCommunaute.SelectedIndex;

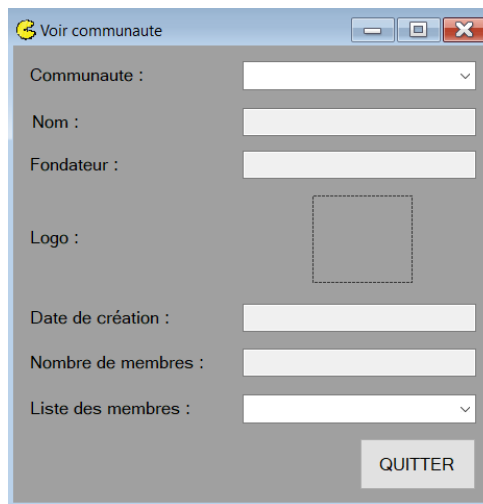
//On parcourt tous les joueurs
foreach (Joueur unJoueur in Globale.lesJoueurs)
{
    //Si le joueur appartient à la communauté sélectionnée alors
    if (unJoueur.getCommunaute() == Globale.lesCommunautes.ElementAt(position).getNom())
    {
        //On le retire de la communauté
        unJoueur.setCommunaute("");
    }
}

//On supprime la communauté sélectionnée
Globale.lesCommunautes.Remove(Globale.lesCommunautes.ElementAt(position));
//On informe la réussite
MessageBox.Show("La communauté à bien été supprimée", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
```



3.2.6 Voir une communauté

Avant de rejoindre une communauté ou lorsque l'on est fondateur de celle-ci on peut être amené à vouloir se renseigner sur celle-ci c'est à quoi sert cette form. Cette action est disponible dans le sous-menu *Communauté / Gestion* du menu MDI par l'ouverture de la form **frmVoirCommunaute**. Cette interface contient toutes les informations sur la communauté telles que le fondateur ou la date de création de celle-ci. On retrouve pour cela une comboBox chargé dès l'ouverture de toutes les communautés créées jusqu'à ce jour. Lors de la sélection d'une de celles-ci on recherche dans la collection *lesCommunautes* de la classe **Globale** grâce à une boucle **While + booléen** l'élément sélectionné et on affiche les informations.



```
int position = cbCommunaute.SelectedIndex; //On recupère l'index de l'élément choisi
int idx = 0; //Variable utile au parcours
bool trouve = false; //Variable utile à l'affichage

//Tant que la collection n'as pas été parcourue totalement et que trouve est faux alors
while (idx < Globale.lesCommunautes.Count && !trouve)
{
    //Si l'index de l'élément choisi est égal à la variable de parcours alors
    if (position == idx)
    {
        trouve = true; //Trouve passe à true
    }
    else
    {
        idx++; //On regarde l'élément suivant
    }
}

//Si on à trouvé quelque chose alors on affiche
if (trouve)
{
    tbNom.Text = Globale.lesCommunautes.ElementAt(position).getNom();
    tbFondateur.Text = Globale.lesCommunautes.ElementAt(position).getFondateur();
    pbLogo.Image = Globale.lesCommunautes.ElementAt(position).getLogo();
    tbDateCreation.Text = Globale.lesCommunautes.ElementAt(position).getDateCreation().ToString("d");
    tbNbMembres.Text = Globale.lesCommunautes.ElementAt(position).nombreMembre().ToString();

    foreach (Joueur unJoueur in Globale.lesCommunautes.ElementAt(position).getLesMembres())
    {
        cbMembres.Items.Add(unJoueur.getPseudo() + " (" + unJoueur.getNom() + " " + unJoueur.getPrenom() + ")");
    }
}

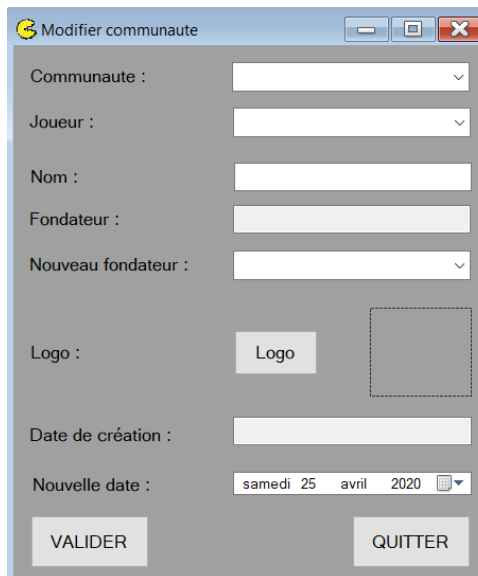
//Sinon on informe l'utilisateur
else
{
    MessageBox.Show("Un problème est survenu avec ce joueur, veuillez réessayer", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```



3.2.7 Modifier la communauté

Pour divers raison on peut vouloir modifier sa communauté afin de changer son nom ou encore son fondateur. Cette action est disponible dans le sous-menu *Communauté / Gestion* du menu MDI par l'ouverture de la form **frmModifCommunaute**. On partira du principe que seul le fondateur de la communauté actuel peut changer les informations de sa communauté. Pour cela on utilise 2 comboBox remplis de toutes les communautés et tous les joueurs pour laisser une ambiguïté et ne pas mettre que les fondateurs (collections *lesCommunautes* et *lesJoueurs* de la classe **Globale**). On les charge à l'aide de 2 **foreach(...)**, les informations resteront bloquées tant que quelque chose ne sera pas sélectionné dans les 2 comboBox.

Les informations de la communauté sélectionnée dans la comboBox contenant celles-ci seront alors affichées dans les textBox correspondantes. Pour la modification on utilisera les mutateurs de la classe communauté. On ne chargera que les joueurs présents dans la communauté grâce à un algorithme **foreach(...)** / **if(...)**, on aura pu utiliser un **foreach(...)** de ce style : `foreach(Joueur unJoueur in Globale.lesCommunautes.ElementAt(cbCommunaute.SelectedIndex).getLesMembres) { ... }` mais j'ai choisi d'utiliser un **foreach(...)** / **if(...)** car je voulais essayer d'utiliser cette méthode plutôt que l'autre afin de manipuler les accesseurs des classes **Communaute** et **Joueur**.



Pour voir le code : cf. [6.3 Code : Modifier la communauté](#)

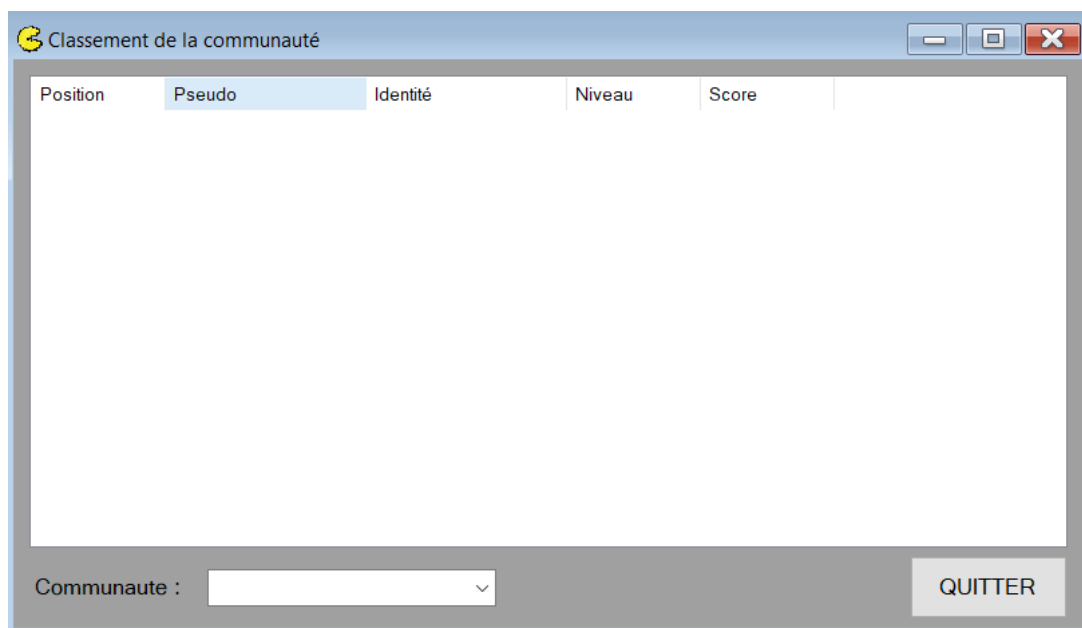
Pour voir le code du bouton logo : cf. [3.2 Algorithmes redondants](#)



3.3 FOURNIR UN CLASSEMENT DES MEMBRES DE LA COMMUNAUTE

Afin de voir sa position par rapport aux autres membres de la communauté, un classement par niveau était imposé comme tâche à réaliser. Cette action est disponible dans le sous-menu *Communaute / ma Communaute* du menu MDI par l'ouverture de la form **frmClassementJoueurCommunaute**. Afin de fournir le classement des membres d'une communauté, il est nécessaire de savoir quelle est la communauté souhaité. Pour cela un comboBox a été implanté dans la form permettant de sélectionner la communauté. Ensuite dès que la communauté a été choisie, le classement par niveau s'affiche dans la listView où sont répertoriées les informations suivantes : Position, Pseudo, Nom Prénom, Niveau et Score. Pour effectuer le tri, on ne tri pas les informations dans la listView mais dans la collection puis on les affiche.

Pour gérer ce tri, on fait appel à une classe auxiliaire, ici c'est la classe **Comparer** (cf. [2.2 Structure des classes](#)). On utilise un **IComparer<...>** de Joueur et une boucle **if(...)** / **else** qui retourne un entier. Si cet entier est 0, c'est que le niveau des 2 joueurs testés sont égaux, si 1, le second joueur est le plus avancé et si c'est -1 alors le premier joueur est le plus avancé. Cela permet de faire fonctionner la méthode **TrierNiveau()** qui nous permet d'appliquer le tri à la collection *lesCommunautes* de la classe **Globale** et donc de mettre celle-ci dans l'ordre décroissant des joueurs par niveau.



Pour voir le code : cf. [6.4 Code : Classement](#)

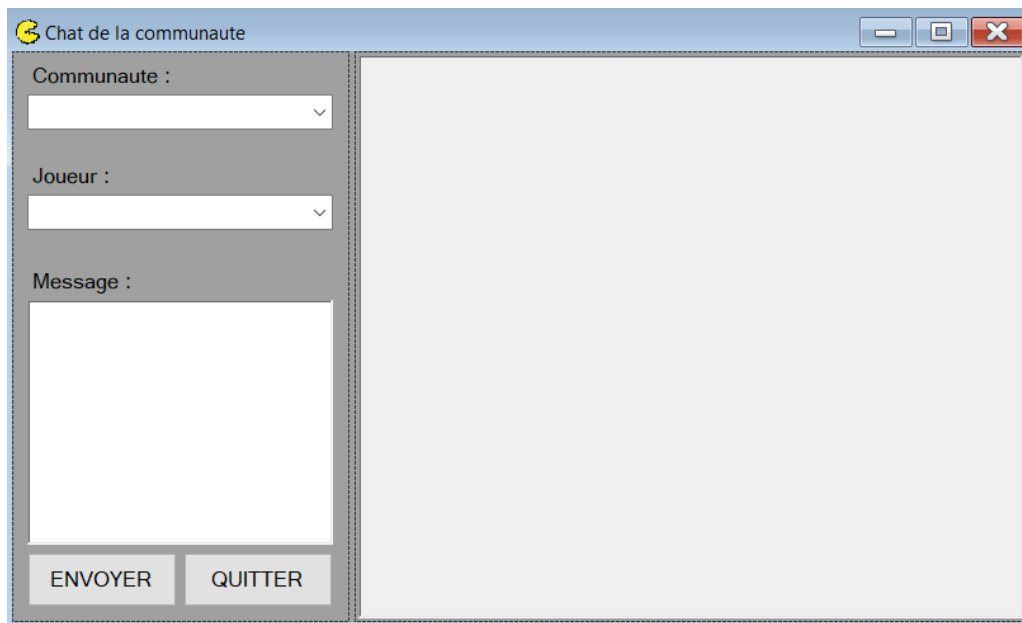


3.4 FACILITER LES ECHANGES ET LE REGROUPEMENT

Afin de pouvoir discuter entre membres d'une même communauté, il est indispensable d'avoir un canal de discussion ou un chat de communauté, ici il a été demandé de réaliser un chat. Cette action est disponible dans le sous-menu *Communaute / ma communaute* du menu MDI par l'ouverture de la form **frmChatCommunaute**. Afin de discuter avec sa communauté, le joueur doit sélectionner dans 2 comboBox différentes la communauté et son profil afin d'écrire son message. Une fois son message écrit il sera instancié un nouvel objet de la classe Message (cf. 2.2 Structures des classes) contenant : l'identité de celui qui l'a envoyé, le contenu de celui-ci et la date et l'heure d'envoi de celui-ci. Une fois cet élément créé il sera affiché dans l'espace de chat sur le côté droit du formulaire d'envoi. Il contiendra les informations stockées sur le message le contenu d'abord puis en dessous l'auteur et la date d'envoi et l'heure d'envoi.

De même que pour **frmModifCommunaute**, seuls les membres de la communauté seront disponibles dans la comboBox permettant de sélectionner le joueur qui va envoyer le message.

Les messages seront dissociés par des espaces de 2 lignes entre chaque. Lors que la communauté sélectionnée dans le comboBox changera, la RichTextBox de chat sera vidée mais les messages seront conservés dans la collection *lesMessages* de la classe **Globale**. Lors du retour sur une communauté qui a déjà envoyé des messages, ceux-ci ne seront plus affichés considérés comme dépassés mais toujours dans la collection. Le chat étant considéré comme un espace de forum et pas de question réponse seulement des messages sont affichés.



Pour voir le code : cf. [6.5 Code : Chat](#)



3.5 MODERER LE FORUM

Afin de rendre les discussions plus agréables et éviter les débordements, il est nécessaire de modérer le forum de chat. Cette action fait partie de la partie précédente et est incorporée dans **frmChatCommunaute**. On cherche ici à éviter toutes les insultes, pour cela plusieurs solutions s'offrent à nous : Ecrire en dur toutes les insultes ou bien créer une collection qui contient celles-ci. Ici la première solution a été retenue. On retrouve donc une procédure dans le code permettant de savoir si le message qui est en train d'être tapé par l'utilisateur contient une insulte écrite en dur. Si c'est le cas un message apparaîtra pour informer l'utilisateur qu'il faut rester courtois et il sera écrit dans son objet qu'une insulte a tentée d'être dite, ce qui pourra par exemple servir si des modalités d'exclusion se mettent en place limitant par exemple le nombre d'insultes tentées à 3, ce qui pourrait conduire à une expulsion du joueur de la communauté. Cela nous servira aussi pour gérer les influenceurs.

```
bool trouve = false;
//Si le texte du controle (remis en min pour identifier même si il y a une maj) contient le mot entre parenthèse alors
if (rtbMessage.Text.ToLower().Contains("putain") || rtbMessage.Text.ToLower().Contains("connard") ||
    rtbMessage.Text.ToLower().Contains("merde") || rtbMessage.Text.ToLower().Contains("fils de pute")
    || rtbMessage.Text.ToLower().Contains("enculé") || rtbMessage.Text.ToLower().Contains("batard")
    || rtbMessage.Text.ToLower().Contains(" salope") || rtbMessage.Text.ToLower().Contains("gourgandine")) //etc...
{
    trouve = true;
}
//Si une insulte à été trouvée
if (trouve)
{
    MessageBox.Show("Veuillez utiliser un langage correct", "Information", MessageBoxButtons.OK, MessageBoxIcon.Hand);
    rtbMessage.Clear();
    //On incrémente le nombre d'injures tentées
    Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex).AugmenterNbInjure();
}
```

```
1 référence
private void rtbMessage_TextChanged(object sender, EventArgs e)
{
    FiltreAObscenite();
    DeblocageBoutonEnvoyer();
}
```



3.6 ANIMER LA COMMUNAUTE

Les personnes qui animent et font vivre la communauté dans la joie de vivre et la bonne ambiance méritent d'être récompensés en ayant leur nom dans une sorte de "mur des trophées". Ces influenceurs se doivent d'être connus. Cette action est disponible dans le sous-menu *Communaute / ma communaute* du menu MDI par l'ouverture de la form **frmInfluenceurCommunaute**. Afin de pouvoir être récompensés, toutes les actions des joueurs se doivent d'être enregistrées. Ici on identifiera 3 type d'influenceurs : Le plus loquace, le plus inactif (ou moins actif), le plus injurieux. Sur la même base que la plupart des autres form les influenceurs seront répertoriés communauté par communauté car ils ne peuvent chatter que dans leur communauté. La sélection de la communauté via un comboBox est donc encore une fois indispensable.

En amont de ce programme, chaque fois qu'un joueur essaiera de poster une insulte ou postera un message les informations seront enregistrés dans leur objet (Incrémentation du nombre de messages postés ou incrémentation du nombre d'insultes tentées). Par la suite des boucles *if(...)* / *else* se mettront en marche permettant d'afficher les influenceurs en dessous de leur récompense.



Pour voir le code : cf. [6.6 Code : Influenceurs](#)



4 JEU D'ESSAI

4.1 LES DIFFERENTS ESSAIS EFFECTUES

<i>Numéro du test</i>	<i>Scénario</i>	<i>Description</i>	<i>Résultat</i>
1	Ouverture	Ouverture de frmNvJoueur	Succès
2	Création	Création d'un nouveau joueur	Succès
3	Fermeture	Fermeture de frmNvJoueur	Succès
4	Ouverture	Ouverture de frmVoirJoueur	Succès
5	Visualisation	Affichage du joueur créé	Succès
6	Vérification	Le joueur est-il dans Globale.lesJoueurs ?	Succès
7	Fermeture	Fermeture de frmVoirJoueur	Succès
8	Ouverture	Ouverture de frmModifMail	Succès
9	Modification	Remplacement du mail du joueur	Succès
10	Fermeture	Fermeture de frmModifMail	Succès
11	Ouverture	Ouverture de frmModifAvatar	Succès
12	Modification	Remplacement de l'avatar du joueur	Succès
13	Fermeture	Fermeture de frmModifAvatar	Succès
14	Ouverture	Ouverture de frmNiveau	Succès
15	Modification	Augmentation du niveau du joueur	Succès
16	Modification	Abaissement du niveau du joueur	Succès
17	Dépassement	Vérification du non dépassement du niveau max	Succès
18	Fermeture	Fermeture de frmNiveau	Succès
19	Ouverture	Ouverture de frmPoint	Succès
20	Modification	Augmentation du score du joueur	Succès
21	Modification	Abaissement du score du joueur	Succès
22	Fermeture	Fermeture de frmPoint	Succès
23	Ouverture	Ouverture frmNvCommunaute	Succès
24	Création	Création d'une communauté	Succès
25	Vérification	La communauté est-elle dans Globale.lesCommunautes ?	Succès
26	Vérification	Le joueur est-il dans la communauté ?	Succès
27	Fermeture	Fermeture de frmNvCommunaute	Succès
28	Ouverture	Ouverture de frmVoirCommunaute	Succès
29	Visualisation	Affichage de la communauté créée	Succès
30	Fermeture	Fermeture de frmVoirCommunaute	Succès
31	Ouverture	Ouverture de frmRejoindreCommunaute	Succès
32	Modification	Ajout d'un joueur dans la communauté	Succès



<i>Numéro du test</i>	<i>Scénario</i>	<i>Description</i>	<i>Résultat</i>
33	Cas particulier	Si le joueur est déjà dans une communauté	Succès
34	Fermeture	Fermeture de frmRejoindreCommunaute	Succès
35	Ouverture	Ouverture frmQuitterCommunaute	Succès
36	Modification	Quitter une communauté	Succès
37	Cas particulier	Si le joueur est le dernier de la communauté	Succès
38	Cas particulier	Si le joueur est fondateur de la communauté	Succès
39	Cas particulier	Si le joueur n'appartient pas à une communauté	Succès
40	Vérification	Mise à vide de la communauté des ex-membres	Succès
41	Fermeture	Fermeture de frmQuitterCommunaute	Succès
42	Ouverture	Ouverture frmSupCommunaute	Succès
43	Modification	Suppression d'une communauté	Succès
44	Vérification	Mise à vide de la communauté des ex-membres	Succès
45	Fermeture	Fermeture de frmSupCommunaute	Succès
46	Ouverture	Ouverture de frmModifCommunaute	Succès
47	Modification	Modification des attributs de la communauté	Succès
48	Fermeture	Fermeture de frmModifCommunaute	Succès
49	Ouverture	Ouverture de frmClassementJoueurCommunaute	Succès
50	Vérification	Vérification du classement	Echec
51	Vérification	Vérification du classement	Echec
52	Vérification	Vérification du classement	Echec
53	Vérification	Vérification du classement	Echec
54	Vérification	Vérification du classement	Echec
55	Vérification	Vérification du classement	Echec
56	Vérification	Vérification du classement	Echec
57	Vérification	Vérification du classement	Succès
58	Fermeture	Fermeture de frmClassementJoueurCommunaute	Succès
59	Ouverture	Ouverture de frmChatCommunaute	Succès
60	Création	Création et envoi de messages	Succès
61	Affichage	Messages bien affichés ?	Succès
62	Fermeture	Fermeture de frmChatCommunaute	Succès
63	Ouverture	Ouverture de frmInfluenceurCommunaute	Succès
64	Vérification	Vérification des données affichées	Echec
65	Vérification	Vérification des données affichées	Succès
66	Fermeture	Fermeture de frmInfluenceurCommunaute	Succès
67	Fermeture	Fermeture du projet	Succès



5 CONCLUSION

5.1 CONCLUSION GENERALE

Ce projet m'a appris de nombreuses notions sur les collections et m'a fait découvrir le *IComparer<...>* ce qui pourrait être utile dans les futurs projets. Ce projet s'est très bien déroulé dans l'ensemble je suis simplement resté bloqué un long moment sur le classement ne connaissant alors pas la méthode citée ci-dessus. J'ai également appris à me conforter avec les collection et méthodes de classe dans leurs utilisations. Ce dossier de programmation a été très utile car ces interfaces sont la base de tout jeux ou applications et il est donc utile de les maîtriser pour bien évoluer dans un environnement de développement en C# utilisant Visual Studio. Ce projet m'a également semblé très utile car plus j'avais plus je me rendais compte qu'il manquait certaines choses comme les cas particuliers des form de communauté.

5.2 AMELIORATIONS POSSIBLES

Les améliorations possibles que je vois sont les suivantes :

- Faire un chat en réseau
- Réafficher les précédents messages lors de la reconnexion au chat
- Pour éviter les comboBox, faire une page de connexion avec un joueur plus présent sans besoin de se chercher dans une comboBox.
- Affiner le classement des joueurs



6 ANNEXES

6.1 CODE : CREER UNE COMMUNAUTE

```
//On regarde si le joueur appartient déjà à une communauté
if (Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex).getCommunaute() != "")
{
    //On informe l'utilisateur
    MessageBox.Show("Ce joueur appartient déjà à une communauté, veuillez la quitter pour en créer une", "Impossible", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
else
{
    //On instancie un nouvel objet de la classe Communaute qui sera ajouté à la collection contenant les communautés
    Globale.lesCommunautes.Add(new Communaute(tbNom.Text, cbJoueur.Text, pbLogo.Image));
    //On ajoute le fondateur à sa communauté
    Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex).setCommunaute(tbNom.Text);

    string nomCommu = Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex).getCommunaute(); //On récupère le nom de la communauté du joueur
    int idx = 0; //Variable utile au parcours
    bool trouve = false; //Variable utile à l'affichage

    //Tant que la collection n'as pas été parcourue totalement et que trouve est faux alors
    while (idx < Globale.lesCommunautes.Count && !trouve)
    {
        //Si l'index de l'élément choisi est égal à la variable de parcours alors
        if (nomCommu == Globale.lesCommunautes.ElementAt(idx).getNom())
        {
            trouve = true; //Trouve passe à true
        }
        else
        {
            idx++; //On regarde l'élément suivant
        }
    }

    //Si on a trouvé la communauté
    if (trouve)
    {
        //On ajoute le membre à la liste des membres
        Globale.lesCommunautes.ElementAt(idx).ajouterMembre(Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex));
    }
    else
    {
        //On informe une erreur
        MessageBox.Show("Un problème est survenu, veuillez réessayer", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    //On informe l'utilisateur que la communauté à bien été créée
    MessageBox.Show("La communauté à bien été créée", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```



6.2 CODE : QUITTER SA COMMUNAUTE

```
int position = cbJoueur.SelectedIndex; //On recupère l'index de l'élément choisi
int idx = 0; //Variable utile au parcours
bool trouve = false; //Variable utile à l'affichage

string nomCommu = Globale.lesJoueurs.ElementAt(position).getCommunaute(); //On recupère le nom de la communaute du joueur
int idx2 = 0; //Variable utile au parcours
bool trouve2 = false; //Variable utile à l'affichage

//Tant que la collection n'as pas été parcourue totalement et que trouve est faux alors
while (idx < Globale.lesJoueurs.Count && !trouve)
{
    //Si l'index de l'élément choisi est égal à la variable de parcours alors
    if (position == idx)
    {
        trouve = true; //Trouve passe à true
    }
    else
    {
        idx++; //On regarde l'élément suivant
    }
}

//Tant que la collection n'as pas été parcourue totalement et que trouve2 est faux alors
while (idx2 < Globale.lesCommunautes.Count && !trouve2)
{
    //Si l'index de l'élément choisi est égal à la variable de parcours alors
    if (nomCommu == Globale.lesCommunautes.ElementAt(idx2).getNom())
    {
        trouve2 = true; //Trouve passe à true
    }
    else
    {
        idx2++; //On regarde l'élément suivant
    }
}

//Si on à trouvé le joueur alors
if (trouve)
{
    //Si le joueur n'a pas de communauté alors
    if (Globale.lesJoueurs.ElementAt(position).getCommunaute() == "")
    {
        //On informe l'utilisateur
        MessageBox.Show("Ce joueur n'a pas de communauté", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        //On met sa communauté à rien
        Globale.lesJoueurs.ElementAt(position).setCommunaute("");
        //On le retire de la liste des membres
        Globale.lesCommunautes.ElementAt(idx2).retirerMembre(Globale.lesJoueurs.ElementAt(position));
        //On informe l'utilisateur
        MessageBox.Show("Vous avez quitté votre communauté", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);

        //On vide son nombre de message postés et d'injures tentées
        Globale.lesJoueurs.ElementAt(position).ResetNbMessage();
        Globale.lesJoueurs.ElementAt(position).ResetNbInjure();
    }
}

//Sinon on informe l'utilisateur
else
{
    MessageBox.Show("Un problème est survenu avec ce joueur, veuillez réessayer", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

//Si on à trouvé la communauté
if (trouve2)
{
    //Si la communauté est vide alors
    if (Globale.lesCommunautes.ElementAt(idx2).nombreMembre() == 0)
    {
        //On la supprime
        Globale.lesCommunautes.Remove(Globale.lesCommunautes.ElementAt(idx2));
        //On informe
        MessageBox.Show("La communauté étant vide elle à été supprimée", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

    //Si le joueur était fondateur de la communauté
    else if (Globale.lesCommunautes.ElementAt(idx2).getFondateur() == Globale.lesJoueurs.ElementAt(position).getPseudo() + " (" +
        Globale.lesJoueurs.ElementAt(position).getNom() + " " +
        Globale.lesJoueurs.ElementAt(position).getPrenom() + ")")
    {
        //On désigne aléatoirement un joueur de la communauté comme nouveau fondateur
        Random rdn = new Random();
        int nouveau = rdn.Next(Globale.lesCommunautes.ElementAt(idx2).nombreMembre());
        Globale.lesCommunautes.ElementAt(idx2).setFondateur(Globale.lesCommunautes.ElementAt(idx2).getLesMembres().ElementAt(nouveau).getPseudo() + " (" +
            Globale.lesCommunautes.ElementAt(idx2).getLesMembres().ElementAt(nouveau).getNom() + " " +
            Globale.lesCommunautes.ElementAt(idx2).getLesMembres().ElementAt(nouveau).getPrenom() + ")");

        //On informe du nouveau fondateur
        MessageBox.Show("Le nouveau fondateur est : " + Globale.lesCommunautes.ElementAt(idx2).getFondateur(), "Information", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}
}
```



6.3 CODE : MODIFIER LA COMMUNAUTE

```
//Si le joueur est le fondateur de la communauté
if (Globale.lesCommunautes.ElementAt(cbCommunaute.SelectedIndex).getFondateur() == Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex).getPseudo() + " (" +
    Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex).getNom() + " " +
    Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex).getPrenom() + ")")
{
    int positionCommu = cbCommunaute.SelectedIndex; //Variable utile pour la recherche dans lesCommunautes
    int positionFonda = cbNvFondateur.SelectedIndex; //Variable utile pour la recherche dans lesJoueurs

    //On parcourt tous les joueurs
    foreach (Joueur unJoueur in Globale.lesJoueurs)
    {
        //Si le joueur appartient à la communauté alors
        if (unJoueur.getCommunaute() == Globale.lesCommunautes.ElementAt(positionCommu).getNom())
        {
            //On change le nom de communauté des joueurs appartenant à celle-ci
            unJoueur.setCommunaute(tbNom.Text);
        }
    }

    Globale.lesCommunautes.ElementAt(positionCommu).setNom(tbNom.Text); //Modification du nom
    Globale.lesCommunautes.ElementAt(positionCommu).setFondateur(Globale.lesJoueurs.ElementAt(positionFonda).getPseudo()); //Modification du créateur
    Globale.lesCommunautes.ElementAt(positionCommu).setLogo(pbLogo.Image); //Modification du logo
    Globale.lesCommunautes.ElementAt(positionCommu).setDateCreation(dtpDateCreation.Value); //Modification de la date de création

    //On affiche les nouvelles informations
    tbNom.Text = Globale.lesCommunautes.ElementAt(positionCommu).getNom();
    cbNvFondateur.Text = Globale.lesCommunautes.ElementAt(positionCommu).getFondateur();
    pbLogo.Image = Globale.lesCommunautes.ElementAt(positionCommu).getLogo();
    tbDateCreation.Text = Globale.lesCommunautes.ElementAt(positionCommu).getDateCreation().ToString("d");

    //On informe de la réussite
    MessageBox.Show("Les informations ont bien été modifiées", "Succès", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
else
{
    //Il n'est pas fondateur donc pas autorisé à modifier
    MessageBox.Show("Vous n'êtes pas autorisé à modifier ces informations", "Refus", MessageBoxButtons.OK, MessageBoxIcon.Hand);
}

int position = cbCommunaute.SelectedIndex; //On récupère l'index de l'élément choisi
int idx = 0; //Variable utile au parcours
bool trouve = false; //Variable utile à l'affichage

//Tant que la collection n'as pas été parcourue totalement et que trouve est faux alors
while (idx < Globale.lesCommunautes.Count && !trouve)
{
    //Si l'index de l'élément choisi est égal à la variable de parcours alors
    if (position == idx)
    {
        trouve = true; //Trouve passe à true
    }
    else
    {
        idx++; //On regarde l'élément suivant
    }
}

//Si on à trouvé quelque chose alors on affiche
if (trouve)
{
    tbNom.Text = Globale.lesCommunautes.ElementAt(position).getNom();
    tbFondateur.Text = Globale.lesCommunautes.ElementAt(position).getFondateur();
    pbLogo.Image = Globale.lesCommunautes.ElementAt(position).getLogo();
    tbDateCreation.Text = Globale.lesCommunautes.ElementAt(position).getDateCreation().ToString("d");
}
//Sinon on informe l'utilisateur
else
{
    MessageBox.Show("Un problème est survenu avec cette communauté, veuillez réessayer", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

//On parcourt tous les joueurs
foreach (Joueur unJoueur in Globale.lesJoueurs)
{
    //Si le joueur appartient à la communauté alors
    if (unJoueur.getCommunaute() == Globale.lesCommunautes.ElementAt(position).getNom())
    {
        //On ajoute le joueur à la comboBox
        cbNvFondateur.Items.Add(unJoueur.getPseudo() + " (" + unJoueur.getNom() + " " + unJoueur.getPrenom() + ")");
    }
}
```



6.4 CODE : CLASSEMENT

```
lvClassement.Items.Clear(); //On vide la listView car on change de communauté
int numero = 1; //Variable utile au renseignement de la position du joueur
Comparer.TrierNiveau();

//On parcourt tous les joueurs
foreach (Joueur unJoueur in Globale.lesJoueurs)
{
    //Si le joueur appartient à la communauté alors
    if (unJoueur.getCommunaute() == Globale.lesCommunautes.ElementAt(cbCommunaute.SelectedIndex).getNom())
    {
        //affichage des joueurs
        ListViewItem ligne = new ListViewItem();
        ligne.Text = numero.ToString();
        ligne.SubItems.Add(unJoueur.getPseudo());
        ligne.SubItems.Add(unJoueur.getNom() + " " + unJoueur.getPrenom());
        ligne.SubItems.Add(unJoueur.getNiveau().ToString());
        ligne.SubItems.Add(unJoueur.getScore().ToString());
        lvClassement.Items.Add(ligne);

        numero++;
    }
}
```

6.5 CODE : CHAT

```
//On copie le message écrit dans la RichTextBox de chat
rtbChat.AppendText(rtbMessage.Text + "\n");

//On change la police pour afficher les informations
rtbChat.AppendText(cbJoueur.Text + ", " + DateTime.Now.ToString("MM/dd/yyyy HH:mm"));

//On evite que les message soient collés
rtbChat.Text += "\n"; //On saute une ligne
rtbChat.Text += "\n"; //On saute une ligne

//On ajoute le message dans la collection LesMessages de la classe Globale
Globale.lesMessages.Add(new Message(cbJoueur.Text, rtbMessage.Text));

//On incrémente le nombre de messages postés du membre
Globale.lesJoueurs.ElementAt(cbJoueur.SelectedIndex).AugmenterNbMessage();

//On remet les informations à zero sauf la communauté et les membres (voir cbCommunaute_SelectedIndexChanged)
rtbMessage.Text = "";
```



6.6 CODE : INFLUENCEURS

```
//Membre le plus loquace
int nbMessageLoquace = -1; //Variable utile à la recherche
string identiteLoquace = ""; //Variable utile à l'affichage

//Membre le plus inactif
int nbMessageInactif = 1000000; //Variable utile à la recherche
string identiteInactif = ""; //Variable utile à l'affichage

//Membre le plus injurieux
int nbMessageInjure = -1; //Variable utile à la recherche
string identiteInjure = ""; //Variable utile à l'affichage

//On parcourt tous les joueurs
foreach (Joueur unJoueur in Globale.lesJoueurs)
{
    //Si le joueur appartient à la communauté alors
    if (unJoueur.getCommunaute() == Globale.lesCommunautes.ElementAt(cbCommunaute.SelectedIndex).getNom())
    {
        //Si le nombre de message posté par le joueur est supérieur à celui renseigné
        if (nbMessageLoquace < unJoueur.getNbMessage())
        {
            //On assimile son nombre de messages posté à la variable de recherche et on stocke son identité
            nbMessageLoquace = unJoueur.getNbMessage();
            identiteLoquace = unJoueur.getPseudo() + " (" + unJoueur.getNom() + " " + unJoueur.getPrenom() + ")";
        }

        //Si le nombre de message posté par le joueur est inférieur à celui renseigné
        else if (nbMessageInactif > unJoueur.getNbMessage())
        {
            //On assimile son nombre de messages posté à la variable de recherche et on stocke son identité
            nbMessageInactif = unJoueur.getNbMessage();
            identiteInactif = unJoueur.getPseudo() + " (" + unJoueur.getNom() + " " + unJoueur.getPrenom() + ")";
        }

        //Si le nombre d'injure tentées par le joueur est supérieur à celui renseigné
        if (nbMessageInjure < unJoueur.getNbInjureEssaye())
        {
            //On assimile son nombre d'injure tenté à la variable de recherche et on stocke son identité
            nbMessageInjure = unJoueur.getNbInjureEssaye();
            identiteInjure = unJoueur.getPseudo() + " (" + unJoueur.getNom() + " " + unJoueur.getPrenom() + ")";
        }
    }
}
```

