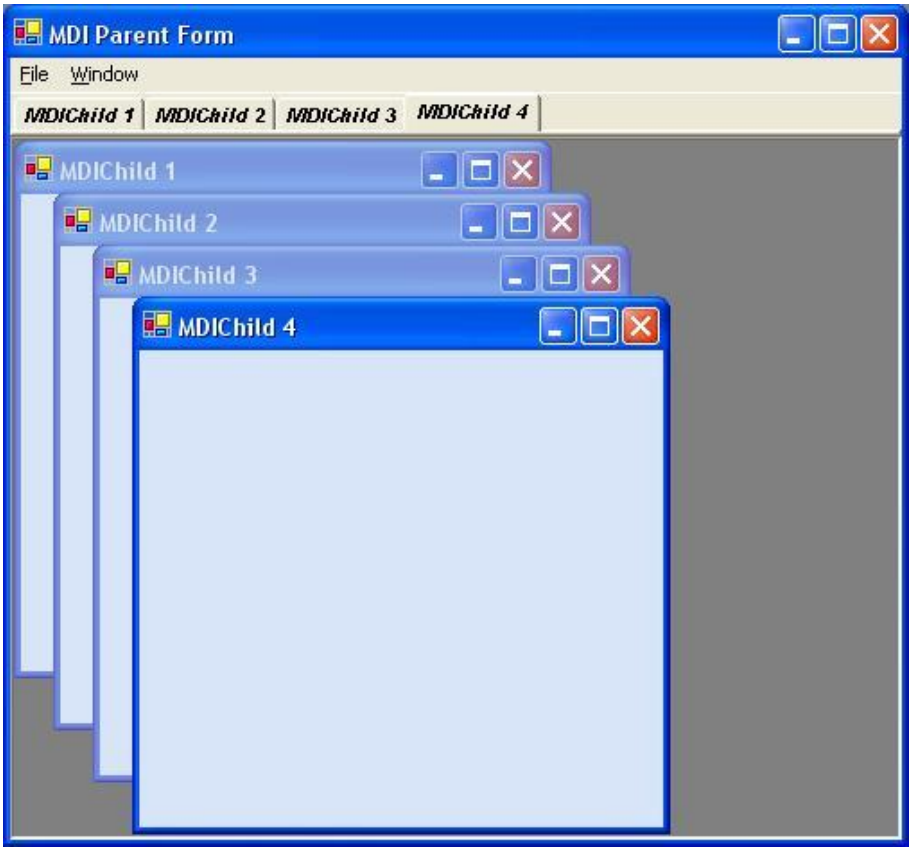



# DOSSIER DE PROGRAMMATION N°2

Menu MDI

RESUME

Développement d'un menu MDI afin d'accueillir une création et gestion d'un joueur

Saint Adjutor, Vernon

# Sommaire :

# 1. Présentation du dossier

## 1.1. Présentation globale

Le dossier de programmation qui nous a été confié par la société BABYLONE, une société de développement spécialisé dans les logiciels de jeux et qui est désireuse de crée une classe Joueur. Celle-ci permettra d'intégrer à n'importe quel jeu, un ou plusieurs joueurs créent à partir de cette classe notamment pour des jeux de plateformes à plusieurs niveaux.

Ce projet sera à développer en langage C# en développant un projet en Windows Form comprenant un menu MDI (Multiple Document Interface) et toutes les formes nécessaires pour satisfaire le besoin du client. Elle devra également être composée de 2 classes : Globale et Joueur dont la structure nous a été donnée en pseudo-code.

Cet interface devra être capable de :

- Instancier un nouveau joueur de la classe Joueur en définissant les données de bases nécessaires à son instanciation (nom, prénom, mail, pseudonyme et avatar). Le type de donnée devra être vérifié et l'adresse mail du type : identification@serveur.ext
- Voir les informations connus sur le Joueur
- Changer le mail (en gardant les critères de validité ci-dessus)
- Changer l'avatar
- Evoluer de niveau via une interface sans dépasser le niveau maximum
- Changer son nombre de points et si celui-ci dépasse le plafond du niveau passer au suivant automatiquement

## 1.2. Liste des compétences

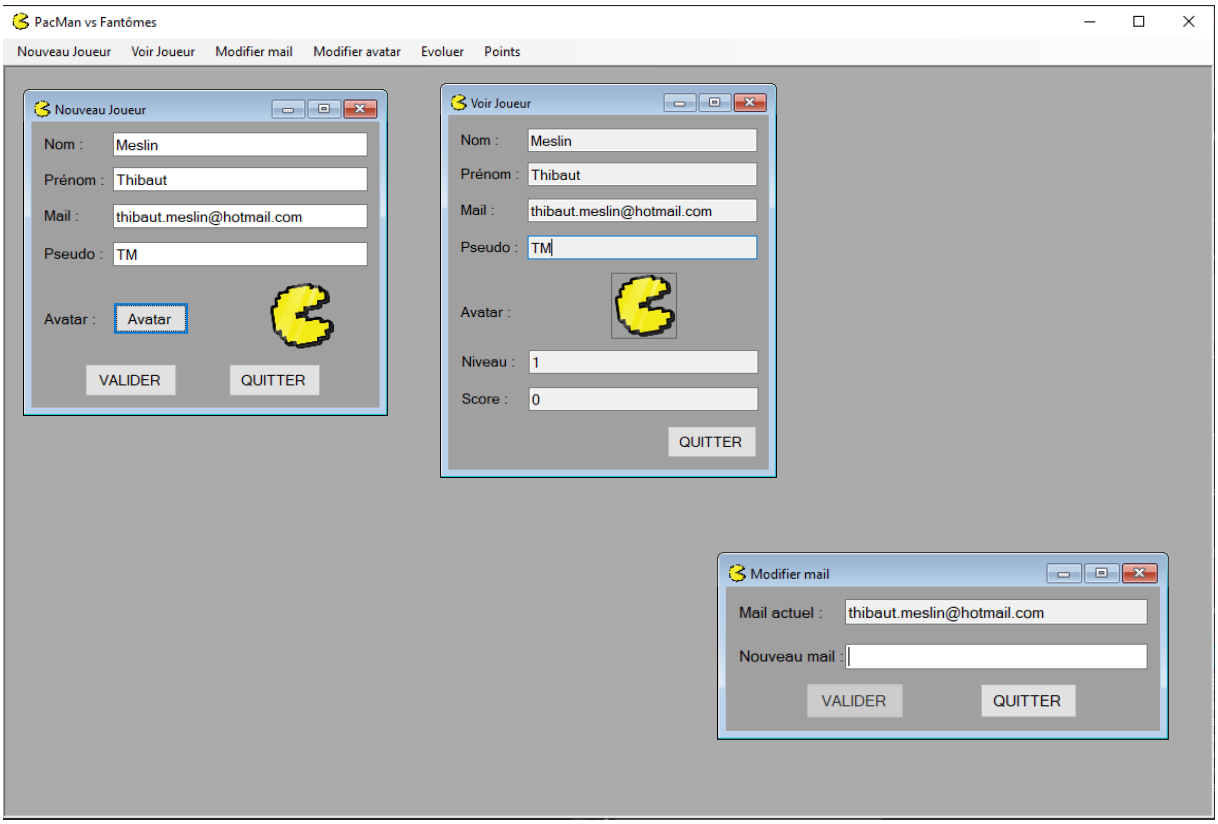
Les compétences associées sont les suivantes :

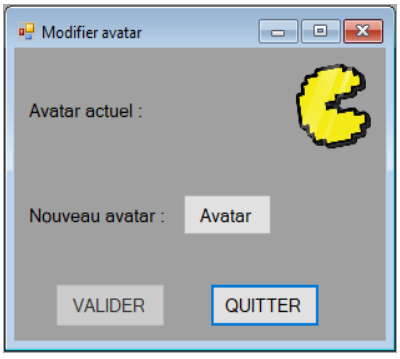
- A4.1.6 Gestion d'environnements de développement et de test
- A4.1.7 Développement, utilisation et adaptation de composants logiciels
- A4.1.8 Réalisation des tests nécessaires à la validation d'éléments adaptés ou développés
- A4.1.9 Rédaction d'une documentation technique

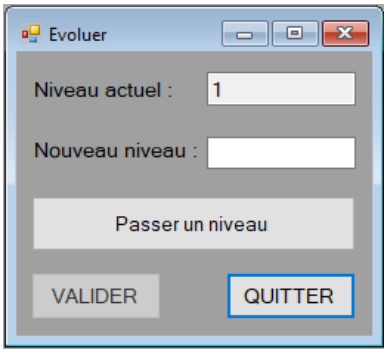
# 2. Structures de données

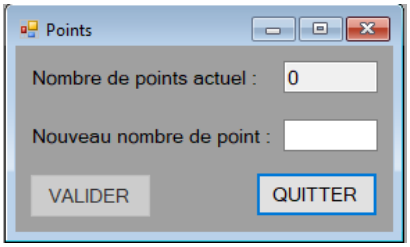
## 2.1. Structure globale

Mon projet se compose des deux classes traduites en langage C# et de 7 form, une servant de menu principale étant donc la form PARENT des autres. Chacune des autres form ont une fonction spécifique dans celles listées au-dessus. Le menu principal est un conteneur MDI possédant une barre de navigation supérieure permettant d'ouvrir les autres menus.



## 2.2. Structure globale

Mon projet dispose de 2 classes une classe Globale contenant le Joueur utilisé pour le test et les variables pour le nombre de niveau et le score maximum déclarer en static integer.

```
32 références
class Globale
{
    public static int nbMaxNiveaux = 5;     //Nombre maximum de niveaux
    public static int plafondScore = 10;    //Score maximum par niveau
    public static Joueur leJoueurTest;      //Joueur du test
}
```

La seconde classe est la classe Joueur servant de base à l'instanciation des futurs joueurs du jeu et contient donc toutes les informations sur ceux-ci (nom, prénom, mail, …).

```
3 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
```

Son constructeur avait des informations à respecter. Le joueur devait voir son niveau instancié à 1 et son score instancié lui à 0. L'avatar choisi par le joueur devait-être stocké dans l'objet créé par cette classe, il est donc présent ici reprenant une structure de classe Image.

```
//Constructeur
1 référence
public Joueur(string leNom, string lePrenom, string leMail, string lePseudo, Image lAvatar)
{
    this.nom = leNom;
    this.prenom = lePrenom;
    this.mail = leMail;
    this.pseudo = lePseudo;
    this.avatar = lAvatar;
    this.niveau = 1;        //On initialise le niveau à 1
    this.score = 0;         //On initialise le score à 0
}
```

# 3. Algorithmes utilisés

## 3.1. Menu global

Le menu global est assez simple et est seulement présent afin d'être le PARENT des autres form, c'est-à-dire de les contenir. Celui-ci avant de devenir un menu MDI était simplement une form. Afin de le faire devenir un menu MDI, la propriété « isMdiContainer » activé, c'est-à-dire isMdiContainer = true.

Le menu comme son nom l'indique permet d'ouvrir les autres form qui lui sont rattachées et lors de l'appui sur un bouton du menu, de les ouvrir. Pour cela il faut instancier un nouvel objet de la form désirée, définir le form du menu comme la form PARENT de celle-ci et ensuite on a simplement à ouvrir l'instance crée plus tôt.

```
private void nouveauJoueurToolStripMenuItem_Click(object sender, EventArgs e)
{
    frmNvJoueur laForm = new frmNvJoueur(); //Instanciation d'un nouvel objet de classe frmNvJoueur
    laForm.MdiParent = this;                //Le parent de cet objet est frmMenu
    laForm.Show();                          //Ouverture de frmNvJoueur
}

1 référence
private void voirJoueurToolStripMenuItem_Click(object sender, EventArgs e)
{
    frmVoirJoueur laForm = new frmVoirJoueur(); //Instanciation d'un nouvel objet de classe frmVoirJoueur
    laForm.MdiParent = this;                    //Le parent de cet objet est frmMenu
    laForm.Show();                              //Ouverture de frmVoirJoueur
}

1 référence
private void modifierMailToolStripMenuItem_Click(object sender, EventArgs e)
{
    frmModifMail laForm = new frmModifMail();   //Instanciation d'un nouvel objet de classe frmModifMail
    laForm.MdiParent = this;                    //Le parent de cet objet est frmMenu
    laForm.Show();                              //Ouverture de frmModifMail
}

1 référence
private void modifierAvatarToolStripMenuItem_Click(object sender, EventArgs e)
{
    frmModifAvatar laForm = new frmModifAvatar();   //Instanciation d'un nouvel objet de classe frmModifAvatar
    laForm.MdiParent = this;                        //Le parent de cet objet est frmMenu
    laForm.Show();                                  //Ouverture de frmModifAvatar
}
private void evoluerToolStripMenuItem_Click(object sender, EventArgs e)
{
    frmEvoluer laForm = new frmEvoluer();   //Instanciation d'un nouvel objet de classe frmEvoluer
    laForm.MdiParent = this;                //Le parent de cet objet est frmMenu
    laForm.Show();                          //Ouverture de frmEvoluer
}

1 référence
private void pointToolStripMenuItem_Click(object sender, EventArgs e)
{
    frmPoints laForm = new frmPoints(); //Instanciation d'un nouvel objet de classe frmPoints
    laForm.MdiParent = this;            //Le parent de cet objet est frmMenu
    laForm.Show();                      //Ouverture de frmPoints
}
```

## 3.2. Nouveau joueur

Afin de créer un nouveau joueur, ici on instanciera seulement le joueur test présent dans la classe Globale, il faut instancier un nouvel objet de la classe Joueur. Mais avant cela le cahier des charges précises que les informations doivent toutes être présentes et que l'adresse mail doit suivre certaines contraintes définies plus haut. Pour cela, la méthode DeblocageBoutonValider a été créée. Elle sera recopiée et adaptée dans toutes les form pour vérifier qu'une information a été rentrée dans l'élément nécessaire. Ici on vérifie simplement que les textbox ne sont pas vident grâce à un « != « » » et si le mail contient un point et un arobase grâce à un .Contain(« . ») et un .Contain(« @ »). On appelle ensuite la méthode à chaque évènement Text_Changed afin de vérifier chaque saisie de la part du joueur.

```
public void DeblocageBoutonValider()    //On définit si le bouton Valider peut être déverouillé
{
    //Si les textBox ne sont pas vide et que tbMail contient un "." et un "@" et si la PictureBox n'est pas vide alors
    if (tbNom.Text != "" && tbPrenom.Text != "" && pbAvatar.Image != null && tbPseudo.Text != "" && tbMail.Text.Contains("@") == true && tbMail.Text.Contains(".") == true)
    {
        btValider.Enabled = true;   //On débloque le bouton
    }
    else
    {
        btValider.Enabled = false;  //On le vérouille
    }
}
```

Ensuite on instancie leJoueurTest de la classe Globale nous servant de base à notre test afin de vérifier la validité de nos actions et de nos algorithmes. On y stocke les informations rentrées dans les textbox par le joueur et l'image entrée de son pc (cf. 3.5 Modifier l'avatar pour le code d'ouverture du fichier).

```
private void btValider_Click(object sender, EventArgs e)
{
    //On instance leJoueurTest présent dans la classe Globale avec le constructeur de la classe Joueur
    Globale.leJoueurTest = new Joueur(tbNom.Text, tbPrenom.Text, tbMail.Text, tbPseudo.Text, pbAvatar.Image);

    //On vide ensuite les textBox et la PictureBox
    tbNom.Clear();
    tbPrenom.Clear();
    tbMail.Clear();
    tbPseudo.Clear();
    pbAvatar.Image = null;
}
```

## 3.3. Voir Joueur

Le code de cette form est très simple. En effet le seul besoin de cette form est d'afficher les informations du joueur, or ici nous ne traitons que du joueur test de la classe Globale donc pas besoin de comboBox et de gestion d'index ici. Pour afficher les informations nous avons seulement à appeler nos getters / accesseurs présent dans la classe Joueur à l'objet leJoueurTest et à assigner les informations relevées à la textbox correspondante ou bien à la PictureBox pour l'image.

```
private void frmVoirJoueur_Load(object sender, EventArgs e)
{
    tbNom.Text = Globale.leJoueurTest.getNom();                    //On affiche le nom stocké dans leJoueurTest
    tbPrenom.Text = Globale.leJoueurTest.getPrenom();              //On affiche le prenom stocké dans leJoueurTest
    tbMail.Text = Globale.leJoueurTest.getMail();                  //On affiche le mail stocké dans leJoueurTest
    tbPseudo.Text = Globale.leJoueurTest.getPseudo();              //On affiche le pseudo stocké dans leJoueurTest
    pbAvatar.Image = Globale.leJoueurTest.getAvatar();             //On affiche l'avatar stocké dans leJoueurTest
    tbNiveau.Text = Globale.leJoueurTest.getNiveau().ToString();   //On affiche le niveau stocké dans leJoueurTest
    tbScore.Text = Globale.leJoueurTest.getScore().ToString();     //On affiche le score stocké dans leJoueurTest
}
```

```
1 référence
public string getNom() { return nom; }          //Retourner le nom du joueur
1 référence
public string getPrenom() { return prenom; }    //Retourner le prénom du joueur
3 références
public string getMail() { return mail; }        //Retourner le mail du joueur
1 référence
public string getPseudo() { return pseudo; }    //Retourner le pseudo du joueur
3 références
public Image getAvatar() { return avatar; }     //Retourner l'avatar du joueur
6 références
public int getNiveau() { return niveau; }       //Retourner le niveau du joueur
4 références
public int getScore() { return score; }         //Retourner le score du joueur
```

## 3.4. Modifier le mail

Modifier le mail requière de vérifier une nouvelle fois la validité de l'adresse mail en entré afin de voir si elle répond aux critères de validité fournis par la société BABYLONE. Pour cela on réutilise le .Contain() et on y réaffecte le point et l'arobase. Ensuite on fait appel à un setter / mutateur de la classe Joueur afin de remplacer l'ancienne adresse par celle entrée maintenant puis on met à jour l'affichage afin de montrer la nouvelle adresse dans le champ « mail actuel ». Si toute fois le mail entré ne répondait pas aux critères, on informera l'utilisateur via une messageBox et remettrai à vide la textBox d'entrée.

```
if (tbNvMail.Text.Contains("@") == true && tbNvMail.Text.Contains(".") == true) //Si le nouveau mail contient un "." et un "@" alors il est valide donc
{
    Globale.leJoueurTest.setMail(tbNvMail.Text);        //On stocke le nouveau mail à la place de l'ancien
    tbActMail.Text = Globale.leJoueurTest.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 sont adresse email n'est pas valide
    MessageBox.Show("Le mail ne répond pas aux critères de validités, veuillez vérifier", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Pour le déblocage du bouton valider, on teste simplement s'il y a quelque chose dans la zone d'écriture.

```
public void DeblocageBoutonValider()    //On définit si le bouton Valider peut être déverouillé
{
    if (tbNvMail.Text != "")  //S'il y a du texte dans la PictureBox du nouveau mail alors
    {
        btValider.Enabled = true;   //On débloque le bouton Valider
    }
    else
    {
        btValider.Enabled = false;  //Il reste bloqué
    }
}
```

```
1 référence
public void setMail(string leMail) { this.mail = leMail; }          //Définir le nouveau mail du joueur
```

## 3.5. Modifier l'avatar

Pour modifier l'avatar on reprend l'algorithme de la création d'un nouveau joueur pour l'ouverture du fichier. On va ouvrir par le clic sur le bouton « avatar » une page de l'ordinateur du joueur afin de lui faire ouvrir une image. Une restriction est appliquée pour que seuls les fichiers images tels que jpeg ou png soit sélectionnable et donc ne pas créer une erreur non voulue. Ensuite si quelque chose a été ouvert par l'utilisateur alors on instancie cet élément via la classe image afin de pouvoir la sauvegarder dans l'instance d'objet du joueur test. Ensuite on utilise simplement un setter / mutateur de la classe Joueur afin de remplacer l'ancien avatar par le nouveau.

```
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
    pbNvAvatar.Image = avatar;                      //On l'affiche dans la PictureBox
}
DeblocageBoutonValider();   //On appelle la méthode pour débloquer le bouton valider
```

```
private void btValider_Click(object sender, EventArgs e)
{
    Globale.leJoueurTest.setAvatar(pbNvAvatar.Image);       //On stocke l'avatar à la place de l'ancien dans leJoueurTest
    pbActAvatar.Image = Globale.leJoueurTest.getAvatar();   //On affiche l'avatar dans la textBox avatar actuel
    pbNvAvatar.Image = null;                                //On vide la PictureBox nouvel avatar
}
```

```
1 référence
public void setAvatar(Image lAvatar) { this.avatar = lAvatar; }     //Définir le nouvel avatar du joueur
```

## 3.6. Modifier le niveau

Avant de modifier le niveau il faut définir le nombre de niveau que le jeu possède. Pour cela on modifie la valeur de la variable « nbMaxNiveau » dans la classe Globale. Pour la phase de test je l'ai initialisé à 5.

```
public static int nbMaxNiveaux = 5;      //Nombre maximum de niveaux
```

Ensuite pour enfin modifier le niveau, il y a plusieurs choses à respecter. D'abord le niveau saisi par l'utilisateur ne doit pas excéder le nombre maximum de niveau contenus dans le jeu et ensuite un bouton a été créé pour passer directement au niveau supérieur en faisant appel à la méthode de classe Joueur AugmenterNiveau(), celle-ci incrémente simplement le niveau. Ensuite pour le choix du niveau, on fait appel à la méthode de classe Joueur setNiveau().

```
//Si le nombre rentré est inférieur ou égal au nombre maximum de niveau alors
if (int.Parse(tbNvNiveau.Text) <= Globale.nbMaxNiveaux)
{
    Globale.leJoueurTest.setNiveau(int.Parse(tbNvNiveau.Text));     //On stocke la nouvelle valeur du niveau dans LeJoueurTest
    tbActNiveau.Text = Globale.leJoueurTest.getNiveau().ToString(); //On affiche la nouvelle valeur dans la textBox niveau actuel
    tbNvNiveau.Clear();                                             //On vide la textBox nouveau niveau
}
else
{
    //Sinon on informe l'utilisateur que la valeur entrée est incorrecte ou excelle le nombre prévu de niveau
    MessageBox.Show("Le nombre entré dépasse le nombre maximum de niveaux", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
    tbNvNiveau.Clear(); //On vide la textBox nouveau niveau
}
```

```
//si le niveau stocké dans leJoueurTest est inférieur au nombre maximum de niveau alors
if (Globale.leJoueurTest.getNiveau() < Globale.nbMaxNiveaux)
{
    Globale.leJoueurTest.AugmenterNiveau();                         //On appelle la méthode pour augmenter de 1 niveau
    tbActNiveau.Text = Globale.leJoueurTest.getNiveau().ToString(); //On affiche le nouveau niveau
}
else
{
    //Sinon on informe l'utilisateur que la valeur entrée est incorrecte ou excelle le nombre prévu de niveau
    MessageBox.Show("Le nombre entré dépasse le nombre maximum de niveaux", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

```
2 références
public void setNiveau(int leNiveau) { this.niveau = leNiveau; }     //Définir le nouveau niveau du joueur
```

```
1 référence
public void AugmenterNiveau() { this.niveau++; }                    //Incrémenter le niveau
```

## 3.7. Modifier les points

Avant de modifier le nombre de points, il faut comme pour le niveau définir le plafond de score dans la classe Globale définie par la variable « plafondScore ». Ici pour le test le plafond est à 10 points.

```
public static int plafondScore = 10;     //Score maximum par niveau
```

Ensuite pour modifier les points quelques précisions s'imposent. Il est précisé que lorsqu'un nombre de points supérieur au plafond sera entré le jeu passera automatiquement au niveau suivant en remettant son score à 0. Puis vient également le cas lorsque le jeu a atteint son niveau final et donc ne peut plus augmenter en niveau. Pour le premier cas, on compare la valeur entrée au plafond si celle-ci est supérieure ou égale alors on incrémente le niveau et remet le score à 0. Pour le second, si le niveau dépasse le nombre de niveau on affiche un message pour dire que le jeu est terminé. Sinon on change juste via un setter / mutateur le nombre de point de l'instance.

```
//Si le score en entré est supérieur ou égal au score maximum
if (int.Parse(tbNvPoint.Text) >= Globale.plafondScore)
{
    //Si le joueur à atteint le niveau maximum
    if (Globale.leJoueurTest.NiveauMaxAtteint())
    {
        //On l'en informe
        MessageBox.Show("Vous avez atteint le dernier niveau, vous avez gagné", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
        tbNvPoint.Clear();  //On vide la textBox nouveau point
    }
    else
    {
        //Sinon on le fait passer au niveau supérieur et on remet son score à zéro et on l'informe
        MessageBox.Show("Vous avez dépassé le plafond de score vous avez monté d'un niveau", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
        Globale.leJoueurTest.AugmenterScore(0);                              //On remet son score à zéro
        Globale.leJoueurTest.setNiveau(Globale.leJoueurTest.getNiveau() + 1);   //On augmente son niveau de 1
        tbActPoint.Text = Globale.leJoueurTest.getScore().ToString();           //On affiche le nouveau niveau
        tbNvPoint.Clear();                                                      //On vide la textBox nouveau point
    }
}
else
{
    //Sinon on remplace juste les points
    Globale.leJoueurTest.AugmenterScore(int.Parse(tbNvPoint.Text)); //On remplace le score du joueur par le score en entré
    tbActPoint.Text = Globale.leJoueurTest.getScore().ToString();   //On affiche le nouveau score
    tbNvPoint.Clear();                                              //On vide la textBox nouveau point
}
```

```
public bool NiveauMaxAtteint()  //Savoir si le niveau maximum est atteint
{
    if (Globale.nbMaxNiveaux <= niveau) //Si le niveau est supérieur ou égale au niveau maximum alors on retourne vrai sinon on retourne faux
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

# 4. Jeu d'essai

## 4.1. Les différents essais effectués

| *Numéro de test* | *Scénario* | *Description* | *Résultat* |
|---|---|---|---|
| *1* | Ouverture | Ouverture de frmMenu | Succès |
| *2* | Fermeture | Fermeture de frmMenu | Succès |
| *3* | Ouverture | Ouverture de frmNvJoueur | Succès |
| *4* | Fermeture | Fermeture de frmNvJoueur | Succès |
| *5* | Ouverture | Ouverture de frmVoirJoueur | Echec |
| *6* | Ouverture | Ouverture de frmVoirJoueur | Succès |
| *7* | Affichage | Affichage des données de frmNvJoueur dans frmVoirJoueur | Succès |
| *8* | Fermeture | Fermeture de frmVoirJoueur | Succès |
| *9* | Ouverture | Ouverture de frmModifMail | Succès |
| *10* | Affichage | Affichage du mail stocké dans leJoueurTest | Succès |
| *11* | Modification | Modification du mail stocké dans leJoueurTest | Succès |
| *12* | Fermeture | Fermeture de frmModifMail | Succès |
| *13* | Ouverture | Ouverture de frmModifAvatar | Succès |
| *14* | Affichage | Affichage de l'avatar stocké dans leJoueurTest | Succès |
| *15* | Modification | Modification de l'avatar stocké dans leJoueurTest | Succès |
| *16* | Fermeture | Fermeture de frmModifAvatar | Succès |

| *Numéro de test* | *Scénario* | *Description* | *Résultat* |
|---|---|---|---|
| *17* | Ouverture | Ouverture de frmEvoluer | Succès |
| *18* | Affichage | Affichage du niveau stocké dans leJoueurTest | Succès |
| *19* | Modification | Modification du niveau stocké dans leJoueurTest | Echec |
| *20* | Modification | Modification du niveau stocké dans leJoueurTest | Echec |
| *21* | Modification | Modification du niveau stocké dans leJoueurTest | Succès |
| *22* | Fermeture | Fermeture de frmEvoluer | Succès |
| *23* | Ouverture | Ouverture de frmPoints | Succès |
| *24* | Affichage | Affichage du score stocké dans leJoueurTest | Succès |
| *25* | Modification | Modification du score stocké dans leJoueurTest | Echec |
| *26* | Modification | Modification du score stocké dans leJoueurTest | Succès |
| *27* | Fermeture | Fermeture de frmPoints | Succès |
| *28* | Test Finale | Vérification de la totalité du programme | Succès |

# 5. Conclusion

## 5.1. Conclusion globale

Pour conclure ce dossier de programmation, je dirai que le projet était très utile car il peut être utilisé dans une infinité de contexte différents et que chacun trouvera une application qui lui est propre et trouvera moyen de le modifier pour arriver à lui donner la forme et les fonctionnalités qu'il veut. Ce dossier étaient assez facile mais plutôt long à mettre en place mais dans les temps imposé cela restait tout de même très correct et largement suffisant (3 semaines). Je dirais que ce projet m'a apporté beaucoup de choses notamment le savoir de faire ce genre de menu qui comme je l'ai dit est très pratique et modifiable à notre gré.

## 5.2. Améliorations possibles

Ce projet peut être décliné de nombreuses façons et dans de nombreux domaines et pourrait avoir :

- Un tableau de joueur afin d'en gérer un plus grand nombre
- Des comboBox afin de gérer l'affichage des éléments du tableau ci-dessus
- Implémenter un jeu de plateforme pour y jouer
- Automatiser les points afin qu'ils augmentent et donne de la difficulté aux joueurs
- Ajouter de nouveaux ennemis en fonction du niveau
- Ajouter de nouvelles structures en fonction du niveau
- …