



e!Gdiplus

Tutoriel : Programmer un jeu Première partie

Module VBA de gestion d'image

par Thierry GASPHERMENT (arkham46.developpez.com/)

Date de publication : 24/04/08

Dernière mise à jour : 24/04/08

Comment développer un jeu en VBA avec Gdi+.
Première partie : un jeu de tir.

I - Introduction.....	3
II - Spécifications fonctionnelles.....	3
III - Spécifications techniques simplifiées.....	3
IV - Création du formulaire.....	3
V - Création du module de classe cIGdiplus.....	4
VI - Déclaration et initialisation de la classe cIGdiplus.....	4
VII - Dessin du fond étoilé.....	5
VII-A - Création de l'image Gdi+.....	5
VII-B - Dessin des étoiles.....	5
VII-C - Affichage à l'écran.....	5
VIII - La minuterie.....	6
VIII-A - Activation de l'événement "Sur Minuterie".....	6
VIII-B - Choix de l'intervalle de minuterie, la théorie.....	6
VIII-C - Choix de l'intervalle de minuterie, la pratique.....	6
IX - Création, affichage et déplacement du vaisseau.....	7
IX-A - Création d'un module de classe.....	7
IX-B - Déclaration et initialisation de l'objet dans le formulaire.....	7
IX-C - Chargement et affichage de l'image du vaisseau.....	8
IX-D - Gestion de la transparence.....	8
IX-E - Gestion des déplacements.....	9
IX-F - Réinitialisation de l'image à chaque itération.....	10
X - Création, affichage et déplacement des ennemis.....	10
X-A - Création d'un module de classe.....	10
X-B - Déclaration et initialisation de l'objet dans le formulaire.....	11
X-C - Création des ennemis.....	11
X-D - Chargement de l'image des ennemis.....	12
X-E - Affichage et déplacement des ennemis.....	12
XI - Création, affichage et déplacement des missiles.....	12
XI-A - Création d'un module de classe.....	12
XI-B - Déclaration et initialisation de l'objet dans le formulaire.....	13
XI-C - Création des missiles.....	13
XI-D - Affichage et déplacement des missiles.....	14
XII - Gestion des collisions.....	15
XII-A - Création d'une région pour le vaisseau du joueur.....	15
XII-B - Création d'une région pour chaque ennemi.....	16
XII-C - Création d'une région pour chaque missile.....	17
XII-D - Test de collision entre le vaisseau et les ennemis.....	17
XII-E - Test de collision entre les missiles et les ennemis.....	18
XIII - Le son.....	19
XIV - Conclusion.....	19
XV - Téléchargements.....	19

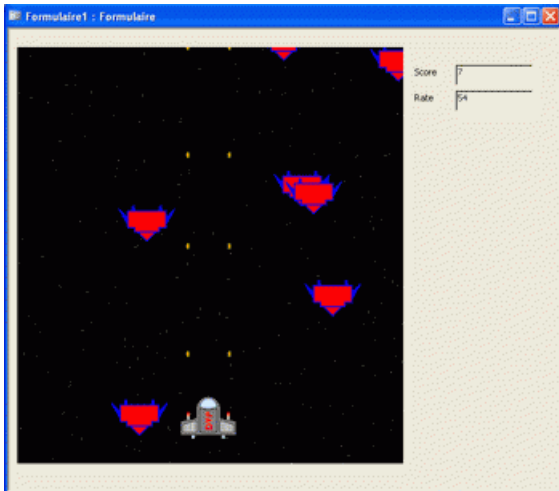
I - Introduction

L'objectif de ce tutoriel est de développer un jeu de tir avec Access.

Pour gérer l'affichage on utilisera une classe spécifique **cIGdiPlus** qui s'appuie sur la librairie Gdi+ de Microsoft.

 **Consultez la documentation et les tutoriels de la classe cIGdiPlus**
N'oubliez pas de télécharger la librairie **gdiplus.dll** si nécessaire.

Voici le résultat en image :



II - Spécifications fonctionnelles

Avant toute chose, nous allons écrire les spécifications fonctionnelles du jeu.

- Un écran de jeu fenêtré (pas en plein écran) pour obtenir une fluidité acceptable
- Le fond d'image est un ciel étoilé statique
- Un vaisseau est dirigé par le joueur sur les deux axes (horizontal et vertical) avec les flèches du clavier
- Le tir du vaisseau du joueur est déclenché avec la touche espace du clavier
- Le tir du vaisseau du joueur est vertical
- Les ennemis apparaissent en haut de l'image et se déplacent vers le bas à une vitesse aléatoire
- Le score est affiché à côté de l'image
- Lorsque le vaisseau du joueur entre en collision avec un ennemi, le jeu s'arrête

III - Spécifications techniques simplifiées

Ci-dessous et en bref, les techniques de base utilisées pour le développement.

Tout sera plus détaillé dans les chapitres suivants.

- Utilisation du timer du formulaire : l'image est mise à jour à chaque événement de ce timer.
- Utilisation de Gdi+ pour le graphisme, avec l'aide de la classe **cIGdiPlus**.
- Création d'un module de classe pour chaque famille d'objet (vaisseau, tir, ennemis, ...).
- Utilisation d'une collection pour les objets multiples (tir, ennemis).
- Le vaisseau du joueur et les ennemis sont des images stockées à l'extérieur de la base de données.
- Les tirs du vaisseau seront dessinés à base de rectangle.

IV - Création du formulaire

On crée un formulaire dans lequel on place :

- Un contrôle image nommé **Img** pour l'affichage du jeu
- Une zone de texte **TxtScore** pour l'affichage du score

La zone de texte est utilisée en affichage uniquement, donc on définit ses propriétés :

- **Activé** => **Non**
- **Vérouillé** => **Oui**

Le formulaire n'est pas utilisé pour afficher des données, donc on peut supprimer les éléments inutiles dans ses propriétés :

- **Afficher sélecteur** => **Non**
- **Boutons de déplacement** => **Non**
- **Diviseur d'enregistrement** => **Non**

V - Création du module de classe clGdiplus

Téléchargez la classe au format texte ([HTTP](#))

Dans l'éditeur VBA :

- 1 - Créez un nouveau module de classe (dans le menu : **insertion** -> **module de classe**).
- 2 - Collez-y le contenu du fichier texte.
- 3 - Sauvegardez le module avec le nom **clGdiPlus**.
- 4 - Compiler (**Débogage** --> **Compiler**)

Attention : le nom sous lequel vous sauvegardez ce module est important.

VI - Déclaration et initialisation de la classe clGdiplus

On va écrire notre code dans le module du formulaire. Cliquez sur **Affichage** --> **Code** pour ouvrir ce module.

Vérifiez que vous avez l'instruction **Option Explicit** en haut du module.

Sinon rajoutez cette ligne, elle impose la déclaration de toutes les variables et évite ainsi les erreurs de saisie dans leur nom.

En-tête de module

```
Option Compare Database
Option Explicit
```

Pour pouvoir utiliser la classe il est nécessaire de la **déclarer** et **l'initialiser**.

La classe est un objet dont le type est le nom sous lequel on a sauvegardé notre module de classe.

Elle se déclare comme n'importe quelle autre variable.

Ensuite on initialise la classe dans l'événement **Sur Ouverture** du formulaire.

Déclaration de la classe

```
Private clGdip As ClGdiPlus ' Classe pour utilisation de gdipplus
```

Initialisation de la classe

```
Private Sub Form_Open(Cancel As Integer)
' Création de la classe à l'ouverture du formulaire
Set clGdip = New clGdiPlus
End Sub
```

Pensez à libérer la classe à la fermeture du formulaire.

(même si celle-ci est libérée automatiquement, je préfère la libérer explicitement).

Dans les propriétés du formulaire, définissez [**Procédure événementielle**] dans l'événement **Sur Fermeture**.

Cliquez sur les trois petits points [...] pour générer l'événement dans le code.

A l'intérieur de la procédure **Form_Close** on va libérer la classe : il suffit de lui donner la valeur **Nothing** si elle n'a pas déjà cette valeur.

Libération de la classe

```
Private Sub Form_Close()
```

Libération de la classe

```
' Libération de la classe à la fermeture du formulaire
If Not clGdip is Nothing Then Set clGdip = Nothing
End Sub
```

VII - Dessin du fond étoilé

VII-A - Création de l'image Gdi+

Pour pouvoir dessiner il faut d'abord créer un bitmap (=une image) avec la fonction **CreateBitmap**.

La taille de ce bitmap doit être exprimée en pixel, donc nous utilisons les fonctions **PointsToPixelsX** et **PointsToPixelsY** pour convertir la taille du contrôle **Img** en pixel.

Le bitmap obtenu est alors de la même taille que le contrôle **Img**.

Notez que nous conservons la taille de l'image dans des variables privés du module afin de les réutiliser plus tard.

Autre remarque : l'image est noire et transparente par défaut; nous remplissons donc l'image de noir opaque pour éviter des soucis de superposition d'image plus tard.

Déclaration en en-tête de module

```
Private gWidth As Long, gHeight As Long ' Taille de l'image en pixel
```

Création du bitmap dans Form_Load

```
' Taille de l'image = taille du contrôle
gWidth = clGdip.PointsToPixelsX(Me.Img.Width)
gHeight = clGdip.PointsToPixelsY(Me.Img.Height)
' Création d'un nouveau bitmap
clGdip.CreateBitmap gWidth, gHeight
' Rempli l'image de noir opaque
clGdip.FillColor vbBlack
```

VII-B - Dessin des étoiles

Les étoiles sont des points dessinés à l'aide de la fonction **DrawPixel**.

La couleur et la position de ces points est aléatoire.

Déclaration à ajouter dans Form_Load

```
Dim lCpt As Long ' Compteur
Dim lCalc As Long ' Variable de calcul
```

Dessin des étoiles après création du bitmap

```
' Dessin des étoiles
Randomize Timer
For lCpt = 1 To 300
    lCalc = 55 + Rnd * 200
    clGdip.DrawPixel Rnd * gWidth, Rnd * gHeight, RGB(lCalc, lCalc, lCalc)
Next
```

VII-C - Affichage à l'écran

Pour l'instant, si on exécute le code, rien ne s'affiche.

On a uniquement dessiné en mémoire.

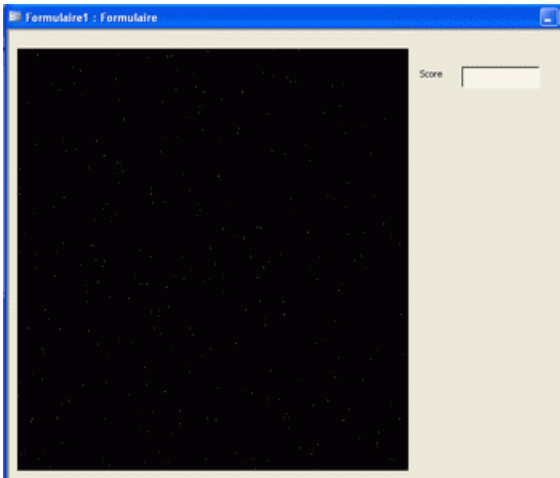
Utilisons la fonction **RepaintControl** pour injecter l'image dans le contrôle **Img**.

Affichage à l'écran, à la suite du code précédent

Affichage à l'écran, à la suite du code précédent

```
' Dessin dans le contrôle
clGdip.RepaintControl Me.Img
```

Afficher maintenant le formulaire pour voir le fond étoilé.



VIII - La minuterie

Pour faire un jeu il faut rafraîchir la position des objets et l'affichage à intervalle régulier.
Pour cette opération nous utilisons le timer du formulaire.

VIII-A - Activation de l'événement "Sur Minuterie"

Celui-ci est accessible dans les propriétés du formulaire dans l'onglet événements:
Définissez [**Procédure événementielle**] dans l'événement **Sur minuterie**.
Puis cliquez sur les trois petits points [...] pour générer l'événement dans le code.
Définissez ensuite l'intervalle de minuterie.

VIII-B - Choix de l'intervalle de minuterie, la théorie.

L'intervalle de minuterie ne doit pas être choisi au hasard.
Il définit le nombre d'image qui sera dessiné par seconde.
Par exemple, en théorie, pour un intervalle de 20 millisecondes :
 $1000/20 = 50$ donc on obtient 50 images par secondes.
Cela peut paraître beaucoup, mais si vous souhaitez déplacer pixel par pixel un objet sur toute la hauteur d'un écran de 800 pixels, il faudra compter :
 $800/50 = 16$ secondes.

VIII-C - Choix de l'intervalle de minuterie, la pratique.

Pour tester notre minuterie, nous allons afficher le nombre d'exécutions par seconde sur le formulaire.
Créer une zone de texte nommée **TxtRate**.
Dans l'événement **Form_Timer** on va compter le nombre d'exécutions par seconde :

Affichage du framerate

```
Private Sub Form_Timer()
' Affiche le framerate
Static sTimer As Double
Static sNb As Long
```

Affichage du framerate

```

Dim lTimer As Double
Dim lInterval As Long
lTimer = Timer
If lTimer - sTimer >= 1 Then
    Me.TxtRate.Value = CLng((sNb + 1) / (lTimer - sTimer))
    sNb = 0
    sTimer = lTimer
Else
    sNb = sNb + 1
End If
End Sub
    
```

Définissez un intervalle de minuterie de 25 ms et exécutez le formulaire.

On s'attend à voir un framerate de $1000/25 = 40$.

Pourtant l'affichage nous indique environ 33 ce qui correspond à un intervalle de 30.

En fait la minuterie n'a pas une résolution de 1 ms.

C'est à dire qu'on ne peut pas définir notre intervalle à la milliseconde près.

On ne peut définir l'intervalle que par pas de 10, 15 ou 55 suivant l'environnement :

- 55 Pour Windows 95 et 98.
- 15 Pour Windows supérieur à 98 et Dual Core.
- 10 Pour Windows supérieur à 98 sans Dual Core.

Choisir un intervalle de 15 ms permet d'avoir 50 images/s sans Dual Core et 66 images/s avec un Dual Core.

C'est cet intervalle de minuterie que nous choisissons pour ce tutoriel.

IX - Création, affichage et déplacement du vaisseau

Comme son titre l'indique, nous allons dans cette section gérer le vaisseau du joueur.

IX-A - Création d'un module de classe

Notre vaisseau est un objet avec des propriétés telles que sa position, sa vitesse, ...

Commençons par créer un module de classe nommée **clShip**.

Inscrivons-y les propriétés évidentes :

- la **position** : X et Y
- la **vitesse** : Speed

Classe clShip

```

Option Compare Database
Option Explicit
Public X As Long, Y As Long ' Position
Public Speed As Long ' Vitesse
    
```

IX-B - Déclaration et initialisation de l'objet dans le formulaire

Comme il n'y a qu'un vaisseau, on lui crée une variable objet **gShip** qu'on initialise à l'ouverture du formulaire et qu'on libère à la fermeture du formulaire.

À l'initialisation, on définit la position du vaisseau et sa vitesse en pixel.

Déclaration en en-tête de formulaire

```

Private gShip As ClShip ' Objet Vaisseau
    
```

Initialisation de l'objet vaisseau Form_Load

```

' Initialisation de l'objet Vaisseau
Set gShip = New ClShip
    
```

Initialisation de l'objet vaisseau Form_Load

```
gShip.X = gWidth / 2 ' Centré horizontalement
gShip.Y = 0.9 * gHeight ' Verticalement à 90% du haut de l'image
gShip.Speed = 5
```

Libération de l'objet vaisseau dans Form_Close

```
' Libération de l'objet vaisseau
Set gShip = Nothing
```

IX-C - Chargement et affichage de l'image du vaisseau

Préparer une image avec votre éditeur de dessin favoris.
Pour ce tutoriel, j'ai simplement utilisé **Paint** et **Microsoft photo editor**.



Eviter le format Jpeg qui compresse l'image, sinon vous ne pourrez plus travailler l'image pour rendre le tour transparent (il faut qu'il soit uni).

Au chargement du formulaire, ajouter l'image du vaisseau dans la classe cIGdiplus.

Vous pouvez également redimensionner cette image.

Chargement et redimensionnement de l'image dans Form_Load

```
' Chargement des images
clGdip.ImageListAdd "ship", CurrentProject.Path & "\img\ship.bmp" ' Chargement du fichier
clGdip.ImageListResize "ship", 80 ' Redimensionne à 80 pixels de largeur
```

L'affichage est géré dans la procédure **Form_Timer**.

Nous utilisons la fonction **DrawImage** pour dessiner le vaisseau sur l'image principale.

Le paramètre **GdipSizeModeAutoSize** indique que l'image est dessinée à sa taille réelle et centrée sur les coordonnées gShip.X et gShip.Y.

Puis la fonction **FastRepaint** redessine l'image du jeu sur le formulaire.

Noter que nous n'injectons pas l'image dans le contrôle avec la fonction **RepaintControl** car ce serait trop lent.

Dessin du vaisseau

```
' Dessine le vaisseau
clGdip.DrawImage "Ship", gShip.X, gShip.Y, , , GdipSizeModeAutoSize
' Dessin l'image sur le formulaire
clGdip.FastRepaint Me.Img
```

IX-D - Gestion de la transparence

Si l'image est dans un format qui ne supporte pas la transparence, le vaisseau est dessiné dans un rectangle plein.

Il faut rendre ce fond transparent :

1 - soit en remplaçant la couleur du fond par une couleur transparente

```
' Chargement des images
clGdip.ImageListAdd "ship", CurrentProject.Path & "\img\ship.bmp" ' Chargement du fichier
clGdip.ImageListResize "ship", 80 ' Redimensionne à 80 pixels de largeur
clGdip.ImageListReplaceColor "ship", vbWhite, vbBlack, , 0 ' Remplace le blanc par du noir transparent
```

2 - soit en définissant la couleur de transparence à l'affichage.

(remarque : l'image étant transformée lors du redimensionnement, cette méthode ne donne pas les meilleurs résultats).

```
' Dessine le vaisseau
clGdip.DrawImage "ship", gShip.X, gShip.Y, , , vbWhite, GdipSizeModeAutoSize
```

3 - soit en sauvegardant l'image dans un format qui supporte la transparence.

Le format PNG donne de bons résultats.

Il n'y a pas alors besoin de gérer la transparence dans le code.

Voici l'image du vaisseau au format PNG avec transparence.



IX-E - Gestion des déplacements

Le déplacement du vaisseau se fait avec les touches fléchées.

Lorsqu'on appuie ou relâche une touche, un événement **KeyDown** puis **KeyUp** est déclenché.

Si on laisse une touche appuyée, des événements **KeyDown** successifs sont déclenchés.

Modifier la position du vaisseau directement sur cet événement rendrait un mouvement très saccadé et fonction de la configuration de la répétition des touches.

Pour avoir un mouvement fluide, on va définir 4 variables booléennes (une par touche) dont on modifiera l'état (Vrai ou Faux) lors des événements **KeyDown** et **KeyUp**.

Déclaration en en-tête de formulaire

```
Private gKeyLeft As Boolean ' Etat flèche gauche
Private gKeyRight As Boolean ' Etat flèche droite
Private gKeyUp As Boolean ' Etat flèche haut
Private gKeyDown As Boolean ' Etat flèche bas
```

Sur touche appuyée

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
Select Case KeyCode
Case 37 ' Gauche
gKeyLeft = True
Case 38 ' Haut
gKeyUp = True
Case 39 ' Droite
gKeyRight = True
Case 40 ' Bas
gKeyDown = True
Case Else
End Select
End Sub
```

Sur touche relâchée

```
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
Select Case KeyCode
Case 37 ' Gauche
gKeyLeft = False
Case 38 ' Haut
gKeyUp = False
Case 39 ' Droite
gKeyRight = False
Case 40 ' Bas
```

Sur touche relâchée

```

gKeyDown = False
Case Else
End Select
End Sub

```

Dans l'événement "Sur minuterie", on vérifie alors l'état des touches grâce à ces variables et on modifie la position du vaisseau en fonction de sa vitesse.

Dans Form_Timer avant l'affichage du vaisseau

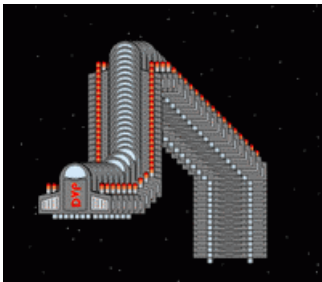
```

' Changement de la position du vaisseau
If gKeyLeft Then gShip.X = gShip.X - gShip.Speed
If gKeyRight Then gShip.X = gShip.X + gShip.Speed
If gKeyUp Then gShip.Y = gShip.Y - gShip.Speed
If gKeyDown Then gShip.Y = gShip.Y + gShip.Speed

```

IX-F - Réinitialisation de l'image à chaque itération

Si vous exécutez le formulaire et déplacez le vaisseau, vous vous apercevez que le vaisseau est dessiné à chaque fois par dessus l'image précédente.



Il faut, avant chaque dessin de l'image dans **Form_Timer**, réinitialiser l'image de fond avant de dessiner les objets dessus.

Dans Form_Load après le dessin des étoiles

```

' Conserve l'image de fond
c1Gdip.KeepImage

```

Dans Form_Timer au début

```

' Rétabli l'image de fond
c1Gdip.ResetImage

```

Avoir rempli l'image de fond de noir opaque à sa création est utile ici sinon le fond serait transparent et ça ne marcherait pas.

Vous pouvez maintenant exécuter le formulaire et utilisez les flèche pour déplacer le vaisseau.

X - Création, affichage et déplacement des ennemis

Comme son titre l'indique, nous allons dans cette section gérer les vaisseaux ennemis.

X-A - Création d'un module de classe

Les ennemis sont, comme le vaisseau du joueur, des objets avec des propriétés telles que leur position, leur vitesse, ... Commençons par créer un module de classe nommée **CIEnnemi**.

Inscrivons-y les propriétés évidentes :

- la **position** : X et Y

- la **vitesse** : Speed

Classe cIEnnemi

```
Option Compare Database
Option Explicit
Public X As Long, Y As Long ' Position
Public Speed As Long ' Vitesse
```

X-B - Déclaration et initialisation de l'objet dans le formulaire

Comme il peut y avoir plusieurs ennemis, on crée une collection qu'on initialise à l'ouverture du formulaire et qu'on libère à la fermeture du formulaire.

Les ennemis seront créés puis insérés dans cette collection.

Déclaration en en-tête de formulaire

```
Private gEnnemis As Collection ' Collection d'ennemis
```

Initialisation de la collection d'ennemis dans Form_Load

```
' Initialisation de la collection d'ennemis
Set gEnnemis = New Collection
```

Libération de la collection d'ennemis dans Form_Close

```
' Libération de la collection d'ennemis
Set gEnnemis = Nothing
```

X-C - Création des ennemis

Les ennemis sont créés en cours de jeu dans la procédure **Form_Timer** puis ajoutés à la collection.

Nous avons donc besoin d'une variable **lEnnemi** de type **cIEnnemi**.

Un ennemi apparaît par exemple toutes les demi-secondes.

Il nous faut donc une variable pour conserver le timer lors de la dernière création d'ennemi.

Déclaration en début de procédure Form_Timer

```
Static sLastEnnemi As Double ' Timer lors de la création du dernier ennemis
Dim lEnnemi As cIEnnemi ' Objet ennemi
```

Si on n'a pas créé d'ennemis depuis plus d'une demi-seconde, on crée un objet ennemi avec sa position et sa vitesse.

L'ennemi apparaît en haut de l'écran, avec une position horizontale et une vitesse aléatoire.

Création d'un ennemi en début de procédure Form_Timer

```
' Nouvel ennemi toute les demi-secondes
If Timer - sLastEnnemi > 0.5 Then
    sLastEnnemi = Timer ' Conserve le timer de la dernière création
    Set lEnnemi = New cIEnnemi ' Nouvel objet
    lEnnemi.X = Rnd * (gWidth) + 1 ' Position horizontale aléatoire (entre 1 et gwidth)
    lEnnemi.Y = 0 ' Position vertical = en haut
    lEnnemi.Speed = Rnd * 9 + 1 ' Vitesse aléatoire (entre 1 et 9 pixel)
    gEnnemis.Add lEnnemi ' Ajoute l'ennemi à la collection
    Set lEnnemi = Nothing ' Libère l'objet
End If
```

Remarque : l'objet est libéré après son ajout à la collection.

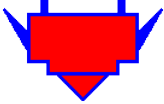
Cela ne signifie pas que l'objet est détruit, car il est toujours contenu dans la collection.

C'est là tout l'avantage d'avoir créé une collection en en-tête de module.

X-D - Chargement de l'image des ennemis

Préparez un fichier image de la même manière que pour le vaisseau puis chargez cette image à l'ouverture du formulaire.

Image utilisée dans ce tutoriel :



Vous pouvez également redimensionner cette image.

Chargement et redimensionnement de l'image dans Form_Load

```
' Chargement des images
clGdip.ImageListAdd "ennemi", CurrentProject.Path & "\\img\ennemi.png" ' Chargement du fichier
clGdip.ImageListResize "ennemi", 80 ' Redimensionne à 80 pixels de largeur
```

X-E - Affichage et déplacement des ennemis

L'affichage et le déplacement des ennemis sont gérés dans la procédure **Form_Timer**.

Comme il y a plusieurs ennemis, il faut faire une boucle sur la collection d'ennemis.

On souhaite également retirer l'ennemi de la collection s'il dépasse le bas de l'écran.

C'est pour cela que le compteur **lCpt** est à rebours; on peut supprimer un élément et continuer la boucle vers les éléments d'indice moins élevé.

Déclaration en début de procédure Form_Timer

```
Dim lCpt As Long ' Compteur
```

Dessin et déplacement des ennemis dans Form_Timer

```
' Ennemis
For lCpt = gEnnemis.Count To 1 Step -1
    Set lEnnemi = gEnnemis.item(lCpt)
    ' Dessine l'ennemi
    clGdip.DrawImage "ennemi", lEnnemi.X, lEnnemi.Y, , , GdipSizeModeAutoSize
    ' Déplace l'ennemi
    lEnnemi.Y = lEnnemi.Y + lEnnemi.Speed
    ' Si le missile arrive en bas de l'écran, on le supprime
    If lEnnemi.Y > gHeight Then
        ' Supprime l'objet
        gEnnemis.Remove lCpt
    End If
Next
```

XI - Création, affichage et déplacement des missiles

Nous allons dans cette section gérer les missiles tirés par le vaisseau du joueur.

XI-A - Création d'un module de classe

Les missiles sont également des objets.

Commençons par créer un module de classe nommée **CIMissile**.

Inscrivons-y les propriétés évidentes :

- la **position** : X et Y
- la **vitesse** : Speed
- la **taille** : Size

Classe CIMissile

```

Option Compare Database
Option Explicit
Public X As Long, Y As Long
Public Speed As Long
Public Size As Long
    
```

XI-B - Déclaration et initialisation de l'objet dans le formulaire

Comme il peut y avoir plusieurs missiles, on crée une collection qu'on initialise à l'ouverture du formulaire et qu'on libère à la fermeture du formulaire.

Les missiles seront créés puis insérés dans cette collection.

Déclaration en en-tête de formulaire

```
Private gMissiles As Collection ' Collection de missiles
```

Initialisation de la collection de missiles dans Form_Load

```
' Initialisation de la collection de missiles
Set gMissiles = New Collection
```

Libération de la collection de missiles dans Form_Close

```
' Libération de la collection de missiles
Set gMissiles = Nothing
```

XI-C - Création des missiles

Les missiles apparaissent lorsqu'on appuie sur la barre d'espace.

On procède de même que les touches fléchées, avec une variable qui contient l'état de la touche espace.

Déclaration en en-tête de formulaire

```
Private gKeySpace As Boolean ' Etat de la touche espace
```

Sur touche appuyée

```
Case 32 ' Espace
    gKeySpace = True
```

Sur touche relâchée

```
Case 32 ' Espace
    gKeySpace = False
```

Les missiles sont créés en cours de jeu dans la procédure **Form_Timer** puis ajoutés à la collection.

Nous avons donc besoin d'une variable **IMissile** de type **CIMissile**.

Pour éviter de créer des missiles trop fréquemment (un à chaque minuterie), on utilise une variable static pour conserver le timer lors de la dernière création de missile.

Déclaration en début de procédure Form_Timer

Déclaration en début de procédure Form_Timer

```
Static sLastMissile As Double ' Timer lors de la création du dernier missile
Dim lMissile As ClMissile ' Objet missile
```

Si la touche espace est appuyée et qu'on n'a pas créé de missile depuis plus de 0.2 seconde, on crée un nouvel objet missile.

Le missile doit apparaître à un emplacement bien défini, en fonction de la position actuelle du vaisseau.

En fait on crée deux missiles car notre vaisseau a deux canons.

Création de missiles en début de procédure Form_Timer

```
' Nouveaux missiles
If gKeySpace Then ' Si espace appuyé
    If Timer - sLastMissile > 0.2 Then ' Si pas de missile depuis plus de 0.2 secondes
        ' Premier missile
        Set lMissile = New ClMissile
        lMissile.X = gShip.X - 27
        lMissile.Y = gShip.Y - 10
        lMissile.Speed = 10
        lMissile.Size = 5
        gMissiles.Add lMissile
        Set lMissile = Nothing
        ' Deuxième missile
        Set lMissile = New ClMissile
        lMissile.X = gShip.X + 23
        lMissile.Y = gShip.Y - 10
        lMissile.Speed = 10
        lMissile.Size = 5
        gMissiles.Add lMissile
        Set lMissile = Nothing
        ' Conserve le timer de la dernière création de missile
        sLastMissile = Timer
    End If
End If
```

XI-D - Affichage et déplacement des missiles

L'affichage et le déplacement des missiles sont gérés dans la procédure **Form_Timer**.

Comme il y a plusieurs missiles, il faut faire une boucle sur la collection de missiles.

On souhaite également retirer le missile de la collection s'il dépasse le haut de l'écran.

C'est pour cela que le compteur **lCpt** est à rebours; on peut supprimer un élément et continuer la boucle vers les éléments d'indice moins élevé.

Pour le dessin du missile, nous n'utilisons pas d'image. Un simple rectangle jaune avec un fond rouge suffit.

Dessin et déplacement des missiles dans Form_Timer après le dessin du vaisseau

```
' Missile
For lCpt = gMissiles.Count To 1 Step -1
    Set lMissile = gMissiles.item(lCpt)
    ' Dessine le missile
    clGdip.DrawRectangle lMissile.X - 1, lMissile.Y - lMissile.Size, lMissile.X + 1,
    lMissile.Y, vbRed, vbYellow, 1
    ' Déplace le missile
    lMissile.Y = lMissile.Y - lMissile.Speed
    ' Si le missile arrive en haut de l'écran, on le supprime
    If lMissile.Y <= 0 Then
        ' Supprime l'objet
        gMissiles.Remove lCpt
    End If
Next
```

Le projet est déjà bien avancé, on peut déplacer le vaisseau du joueur, des ennemis sont affichés, et on peut tirer des missiles.

Il reste un gros travail à faire : gérer les collisions entre les objets.

XII - Gestion des collisions

Nous allons gérer les collisions entre :

- le vaisseau et les ennemis => si collision alors fin du jeu
- les missiles et les ennemis => si collision alors destruction de l'ennemi et du missile

XII-A - Création d'une région pour le vaisseau du joueur

On pourrait utiliser les paramètres **pRegion*** de la fonction **DrawImage** pour créer automatiquement une région constituée des points non transparents de l'image.

Ce serait la solution la plus simple à développer, mais également la plus lente à l'exécution.

Nous allons donc, pour accélérer la vitesse d'exécution du programme, créer des régions polygonales.

Dans la classe **clShip**, nous ajoutons une variable **Polygon** de type **Variant**.

Comme il n'est pas possible de définir un tableau directement dans la déclaration de la variable, nous définissons ce tableau à la création de la classe dans l'événement **Class_Initialize**.

Ajout à la classe clShip

```
Public Polygon As Variant

Private Sub Class_Initialize()
    Polygon = Array(9, 63, 23, 60, 23, 41, 31, 42, 31, 60, 46, 55, 46, 29, 59, 10, 74, 6, 93, 14, 102, 30, 102, 56,
        -
        117, 61, 117, 41, 125, 42, 125, 60, 140, 64, 139, 96, 9, 96, 9, 63)
End Sub
```

Ce tableau est constitué de points en pixels (X1,Y1,X2,Y2, ...) pris sur le fichier image **ship.png**.

Ces points forment un polygone fermé qui détermine les limites du vaisseau.

Après avoir dessiné le vaisseau, nous créons une région à partir de ce polygone.

Pour tester visuellement notre région, la fonction **FillRegion** remplit la région de rouge en semi-transparence.

Dans Form_Timer

```
' Dessine le vaisseau
clGdip.DrawImage "ship", gShip.X, gShip.Y, , , GdipSizeModeAutoSize
' Crée une région polygonale
clGdip.CreateRegionPolygon "ship", gShip.Polygon
' Pour test, remplit la région
clGdip.FillRegion "ship", vbRed, , , 150
```

Affichez le formulaire : vous visualisez en rouge la région créée.

Cette région ne suit pas le déplacement du vaisseau et est plus grande que l'image du vaisseau.

Le problème de taille est dû au fait qu'on a redimensionné l'image après l'avoir chargée.

Les coordonnées prise sur l'image avant redimensionnement doivent être corrigée.

Le facteur de correction est, pour l'image de ce tutoriel : 80 / 156

- 80 taille avant redimensionnement

- 156 taille après redimensionnement

Correction de taille

```
' Crée une région polygonale
clGdip.CreateRegionPolygon "ship", gShip.Polygon
' Redimensionne la région
clGdip.ScaleRegion "ship", 80 / 156, 80 / 156
' Pour test, remplit la région
clGdip.FillRegion "ship", vbRed, , , 150
```

Pour repositionner la région, il faut la déplacer en fonction de la position et de la taille de l'image du vaisseau.

Correction de position

```

' Crée une région polygonale
clGdip.CreateRegionPolygon "ship", gShip.Polygon
' Redimensionne la région
clGdip.ScaleRegion "ship", 80 / 156, 80 / 156
' Déplace la région en fonction de la position du vaisseau
clGdip.TranslateRegion "ship", gShip.X - clGdip.ImageListWidth("ship") / 2, gShip.Y -
clGdip.ImageListHeight("ship") / 2
' Pour test, rempli la région
clGdip.FillRegion "ship", vbRed, , , 150
    
```

Affichez le formulaire : vous voyez maintenant la région en rouge se déplacer avec le vaisseau. Vous pouvez retirer maintenant la ligne de code de remplissage de la région.

XII-B - Création d'une région pour chaque ennemi

Comme pour le vaisseau du joueur, on utilise des régions polygonales.

Pour les ennemis cependant, chacun doit pouvoir être identifié de manière unique.

Chaque instance d'une classe possédant un pointeur unique, nous allons l'utiliser.

Dans la classe **CIEnnemi**, nous ajoutons une variable **Polygon** de type **Variant** et sa définition dans l'événement **Class_Initialize**.

Ajout à la classe CIEnnemi

```

Public Polygon As Variant
Public Id as long

Private Sub Class_Initialize()
' Identifiant de l'objet
Id = ObjPtr(Me)
' Polygon pour région
Polygon = Array(4, 12, 23, 30, 24, 16, 34, 16, 34, 6, 42, 6, 42, 16, 98, 16, 98, 6, 105, 6, 105, 16, 116, 16,
-
116, 30, 136, 12, 117, 47, 117, 58, 101, 58, 100, 68, 91, 69, 70, 89, 48, 69, 40, 69, 39, 59, 23, 58, 24, 48, 4
End Sub
    
```

Ce tableau est constitué de points en pixels (X1,Y1,X2,Y2, ...) pris sur le fichier image **ennemi.png**.

Ces points forment un polygone fermé qui détermine les limites d'un ennemi.

Après avoir dessiné chaque ennemi, nous créons une région à partir de ce polygone.

Les corrections de position et de taille adoptent le même principe que pour le vaisseau.

Le nom utilisé pour la région est l'Id de l'objet; ainsi nous créons une région distincte par objet.

Pour tester visuellement notre région, la fonction **FillRegion** remplit la région de rouge en semi-transparence.

Dans Form_Timer

```

' Dessine l'ennemi
clGdip.DrawImage "ennemi", lEnnemi.X, lEnnemi.Y, , , , GdipSizeModeAutoSize
' Crée une région polygonale
clGdip.CreateRegionPolygon lEnnemi.Id, lEnnemi.Polygon
' Redimensionne la région
clGdip.ScaleRegion lEnnemi.Id, 80 / 156, 80 / 156
' Déplace la région en fonction de la position du vaisseau
clGdip.TranslateRegion lEnnemi.Id, lEnnemi.X - clGdip.ImageListWidth("ennemi") / 2, lEnnemi.Y -
clGdip.ImageListHeight("ennemi") / 2
' Pour test, rempli la région
clGdip.FillRegion lEnnemi.Id, vbRed, , , 150
    
```

Affichez le formulaire : vous voyez maintenant des régions en rouge se déplacer avec chaque ennemi.

Vous pouvez retirer maintenant la ligne de code de remplissage de la région.

Mais on n'en a pas encore terminé avec les régions des ennemis.

Lorsqu'un ennemi disparaît en bas de l'écran, on supprime l'objet de la collection mais sa région n'est pas supprimée automatiquement.

Avant de retirer l'objet de la collection (et donc de le détruire) on va supprimer sa région.

Dans Form_Timer

```
' Supprime la région
clGdip.DeleteRegion lEnnemi.Id
' Supprime l'objet
gEnnemis.Remove lCpt
```

XII-C - Création d'une région pour chaque missile

Pour les missiles les régions sont rectangulaires.

Il suffit d'ajouter le nom de la région à créer en paramètre de la fonction **DrawRectangle**.

On ajoute, comme pour les ennemis, un identifiant dans la classe **clMissile** utilisé pour distinguer les régions.

Et n'oublions pas de supprimer la région d'un missile lorsqu'on le détruit.

Ajout à la classe clMissile

```
Public Id As Long

Private Sub Class_Initialize()
    Id = ObjPtr(Me)
End Sub
```

Modification dans Form_Timer pour création de la région

```
' Dessine le missile et crée sa région
clGdip.DrawRectangle lMissile.X - 1, lMissile.Y - lMissile.Size, lMissile.X + 1,
lMissile.Y, vbRed, vbYellow, 1, , , lMissile.Id
```

Modification dans Form_Timer pour suppression de la région

```
If lMissile.Y <= 0 Then
    ' Supprime la région
    clGdip.DeleteRegion lMissile.Id
    ' Supprime l'objet
    gMissiles.Remove lCpt
End If
```

Toutes les régions sont créées. On peut maintenant tester les collisions.

XII-D - Test de collision entre le vaisseau et les ennemis

Pour tester l'intersection entre deux régions, nous utilisons la fonction **RegionsIntersect**

Après chaque création de région d'un ennemi, on teste si la région de cet ennemi et la région du vaisseau ont une intersection.

Si le vaisseau entre en collision avec un ennemi, on positionne un flag à Vrai, puis en fin de procédure on teste ce flag pour afficher le texte de fin de jeu.

Ajout de déclaration dans Form_Timer

```
Dim lGameOver As Boolean ' Flag pour fin de jeu
```

Test collision vaisseau avec ennemi

```
[...]
' Déplace la région en fonction de la position du vaisseau
```

Test collision vaisseau avec ennemi

```

    clGdip.TranslateRegion lEnnemi.Id, lEnnemi.X - clGdip.ImageListWidth("ennemi") / 2, lEnnemi.Y -
    clGdip.ImageListHeight("ennemi") / 2
    ' Test collision avec vaisseau
    If clGdip.RegionsIntersect("ship", lEnnemi.Id) Then
        lGameOver = True
    End If
    
```

Fin de jeu : en fin de procédure Form_Timer

```

' Test game over
If lGameOver Then
    ' Stoppe la minuterie
    Me.TimerInterval = 0
    ' Affiche un texte du fin de jeu
    ' On affiche d'abord le texte en noir et décalé pour un effet d'ombrage du texte
    clGdip.DrawText "GAME OVER", 65, "Arial", 0 + 3, 0 + 3, gWidth + 3,
    gHeight + 3, , , vbWhite, 150, , , , , , True
    clGdip.DrawText "GAME OVER", 65, "Arial", 0, 0, gWidth, gHeight, , , vbRed, , , , , , True
    ' Dessine l'image de manière permanente dans le contrôle
    clGdip.RepaintControl Me.Img
End If
    
```

Affichez le formulaire : si le vaisseau touche un ennemi, le jeu s'arrête.

XII-E - Test de collision entre les missiles et les ennemis

Pour tester l'intersection entre deux régions, nous utilisons la fonction **RegionsIntersect**

Après chaque création de région d'un missile, on teste si la région de ce missile a une intersection avec un ennemi.

Pour chaque missile, il nous faudra faire une boucle sur les ennemis; on déclare donc un deuxième compteur.

Si le missile entre en collision avec un ennemi, on supprime le missile et l'ennemi.

On gère également le score qui est incrémenté à chaque ennemi détruit. Il faut penser à initialiser le score.

Initialisation du score dans Form_Load

```

' Initialisation du score
TxtScore.Value = 0
    
```

Ajout de déclaration dans Form_Timer

```

Dim lCpt2 As Long ' Compteur
    
```

Test collision missile avec ennemi dans Form_Timer

```

[...]
' Dessine le missile et crée sa région
clGdip.DrawRectangle lMissile.X - 1, lMissile.Y - lMissile.Size, lMissile.X + 1,
lMissile.Y, vbRed, vbYellow, 1, , , lMissile.Id
' Collision missile avec ennemi
For lCpt2 = gEnnemis.Count To 1 Step -1
    Set lEnnemi = gEnnemis.item(lCpt2)
' Test de l'intersection entre le missile et l'ennemi
    If clGdip.RegionsIntersect(lMissile.Id, lEnnemi.Id) Then
' Si intersection, on supprime l'ennemi et sa région
        clGdip.DeleteRegion lEnnemi.Id
        gEnnemis.Remove lCpt2
' Puis on supprime le missile et sa région
        clGdip.DeleteRegion lMissile.Id
        gMissiles.Remove lCpt
' Et on incrémente le score de 1
        TxtScore.Value = TxtScore.Value + 1
' On ne peut détruire qu'un seul ennemi avec un missile donc on quitte la boucle
        Exit For
    End If
Next
    
```

Attention : si on supprime le missile, on ne peut ensuite plus le déplacer.
Pour éviter une erreur, on teste si la région du missile existe avant de le déplacer.

Ajout test d'existence de missile avant déplacement

```
' Déplace le missile s'il n'a pas été supprimé
If clGdip.RegionExists(lMissile.Id) Then
    ' Déplace le missile
    lMissile.Y = lMissile.Y - lMissile.Speed
    ' Si le missile arrive en haut de l'écran, on le supprime
    If lMissile.Y <= 0 Then
        ' Supprime la région
        clGdip.DeleteRegion lMissile.Id
        ' Supprime l'objet
        gMissiles.Remove lCpt
    End If
End If
```

Affichez le formulaire : les vaisseaux sont détruits par les missiles.

XIII - Le son

Et le son dans tout ça? Le jeu manque de bruitage.

Dans ce tutoriel, je ne détaille pas la possibilité d'intégrer des bruitages.

Sachez qu'il est tout à fait possible de le faire en VBA :

- soit avec l'API **PlaySound**, simple à utiliser mais ne gère qu'un seul son à la fois
- soit avec l'API **mciSendString**, plus complexe mais qui peut jouer plusieurs sons simultanément.

Ci-dessous un exemple d'utilisation de **PlaySound**.

Déclaration de PlaySound

```
Public Declare Function PlaySound Lib "winmm.dll" Alias "sndPlaySoundA" _
    (ByVal lpszSoundName As Any, ByVal uFlags As Long) As Long
```

Exemple d'utilisation de PlaySound

```
PlaySound CurrentProject.Path & "\fx\MonSon.wav", 1
```

XIV - Conclusion

Ce tutoriel est une première approche de ce que l'on peut faire en VBA avec Gdi+.

On a également vu l'utilisation des modules de classe qui nous permettent de structurer proprement notre code.

On pourrait aller bien plus loin en développant des menus, en ajoutant d'autres objets et même animer les images.

La seule limitation est le temps d'exécution qui doit rester inférieur à l'intervalle de minuterie, sous peine de voir apparaître des ralentissements.

Il faut donc ne pas voir trop grand.

 **Consultez le deuxième tutoriel.**

 **Téléchargez les jeux créés avec Gdi+.**

Merci à l'équipe Office de developpez.com pour ses relectures, commentaires et encouragements!

XV - Téléchargements

Téléchargez la classe au format texte (*HTTP*)

Télécharger la base Access de ce tutoriel au format ACCESS 2000 (*HTTP*)