



# e!Gdiplus

## **Tutoriel : Programmer un jeu Deuxième partie**

### **Module VBA de gestion d'image**

par Thierry GASPHERMENT ([arkham46.developpez.com/](http://arkham46.developpez.com/))

Date de publication : 24/04/08

Dernière mise à jour : 24/04/08

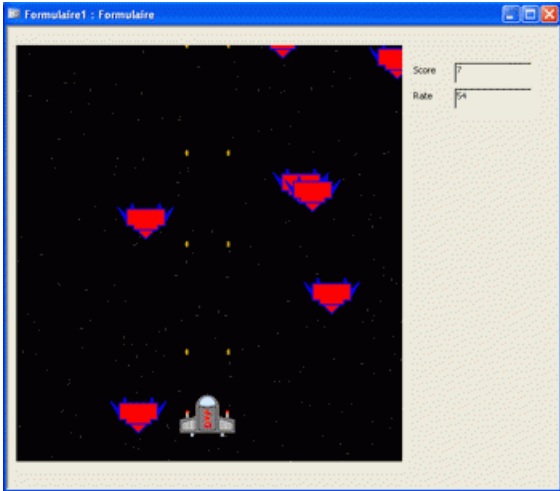
Comment développer un jeu en VBA avec Gdi+.  
Deuxième partie : Boucle de jeu, joystick et sons.  
Utilisation des API multimédia.  
Commentez cet article :

---

I - Introduction.....	3
II - La boucle de jeu.....	3
II-A - Mise en oeuvre de la boucle.....	3
II-B - Exécution de la boucle.....	4
II-C - Ajout d'une fonction d'attente.....	5
II-D - Limitations techniques.....	8
III - Gestions des commandes clavier et joystick.....	8
III-A - Détection du clavier.....	10
III-B - Détection du joystick.....	10
IV - Gestion du son.....	11
V - Conclusion.....	12
VI - Téléchargements.....	12

## I - Introduction

Lors du **premier tutorial**, nous avons commencé le développement d'un petit jeu de tir.



Nous allons, dans cette deuxième partie, apporter quelques améliorations :

- gérer une boucle de jeu (à la place de la minuterie)
- ajouter des sons
- gérer un joystick

**⚠ Consultez la première partie avant de lire ce tutorial**

## II - La boucle de jeu

Nous avons utilisé la minuterie du formulaire pour gérer le jeu du premier tutorial.

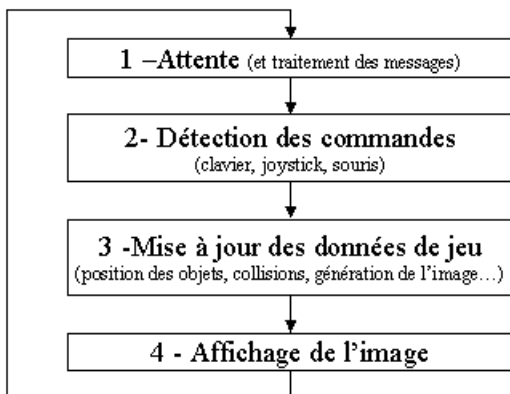
Nous avons également relevé un problème de résolution de cette minuterie, qui est différente selon les configurations, et qui ne permet pas une précision inférieure à 10 ou 15 millisecondes.

Les jeux n'utilisent généralement pas une minuterie de ce type, mais sont gérés par une boucle.

Le terme anglais est **Game Loop**. Si vous recherchez ces mots sur google, vous trouverez de nombreux articles et exemples (souvent en anglais).

Nous allons donc mettre en place une boucle.

Voici l'algorithme de cette boucle :



### II-A - Mise en oeuvre de la boucle

On va créer une boucle simple dans une nouvelle procédure **MainLoop**.

La boucle étant infinie, on utilise une variable booléenne **gStop** pour pouvoir commander la sortie de cette boucle.

#### Déclaration à ajouter en en-tête de formulaire

```
Private gStop As Boolean ' Variable pour sortie de boucle
```

#### Boucle principale

```

'-----
' Boucle principale
'-----
Private Sub MainLoop()
gStop = False
Do
  DoEvents ' Traitement des messages
  If gStop Then Exit Do ' Sortie de la boucle
  UpdateGame ' Mise à jour du jeu
  DisplayGame ' Affichage de l'image
Loop
End Sub

```

#### Arrêt de la boucle à la fermeture du formulaire

```

Private Sub Form_Close()
' Arrêt de la boucle
gStop = True
[...]

```

**UpdateGame** contient la gestion des objets et la génération de l'image de jeu.

On crée donc une nouvelle procédure dans laquelle on déplace le code qui se trouvait dans la procédure **Form\_Timer**.

#### Procédure UpdateGame

```

'-----
' Mise à jour du jeu
'-----
Private Sub UpdateGame()
' Déplacez ici tout le contenu de la procédure Form_Timer
End Sub

```

**DisplayGame** affiche l'image du jeu à l'écran.

On crée donc une nouvelle procédure dans laquelle on place le code d'affichage de l'image.

Retirez cette même ligne de la procédure **UpdateGame** pour ne laisser qu'un seul affichage.

#### Procédure DisplayGame

```

'-----
' Affichage du jeu
'-----
Private Sub DisplayGame()
' Dessine l'image sur le formulaire
cIGdip.FastRepaint Me.Img
End Sub

```

## II-B - Exécution de la boucle

L'exécution de la procédure **MainLoop** fait entrer le programme dans une boucle infinie.

On ne peut donc pas l'exécuter dans l'événement **Sur chargement** du formulaire car il faut que cette procédure de chargement soit menée à son terme.

La solution la plus efficace que j'ai trouvé est d'exécuter la boucle dans la minuterie.

On définit un intervalle de minuterie de 50 millisecondes par exemple. La boucle de jeu démarrera 50 millisecondes après l'ouverture du formulaire.

N'oublions pas de réinitialiser la minuterie après exécution de la boucle afin de ne la lancer qu'une seule fois.

#### Procédure Sur Minuterie

```

'-----
' Procédure exécutée sur minuterie
'-----
Private Sub Form_Timer ()
' Réinitialise la minuterie
Me.TimerInterval = 0
' Exécute la boucle de jeu
MainLoop
End Sub
    
```

Si on exécute le formulaire à ce stade du développement, le jeu se déroule aussi vite que le PC le permet. Il manque une fonction d'attente pour afficher le jeu à une vitesse déterminée.

## II-C - Ajout d'une fonction d'attente

On a vu dans le premier tutoriel que la minuterie avait une résolution de 10 ou 15 ms.

D'autres fonction telles que **GetTickCount** ont la même résolution.

Pour avoir une résolution de 1 ms, on utilise les **Timers Multimedia**.

Ces timers ne sont pas très simples à utiliser.

Ci-dessous, le code d'un module de classe pour gérer l'attente.

#### Classe pour attente avec timer multimedia

```

'*****
'*                               CLASSE POUR ATTENTE                               *
'*****
' Utilisation de timer multimedia : http://msdn2.microsoft.com/en-us/library/ms712704(VS.85).aspx
'*****
' Auteur : Thierry GASPERMENT (Arkham46)
' Le code est libre pour toute utilisation
'*****
' Utiliser la fonction Wait pour temporiser une boucle
' Do
' Wait 10
' ' Traitements à effectuer toutes les 10 ms
' Loop
'*****
Option Explicit

Private Const TIME_PERIODIC = 1
Private Const TIME_CALLBACK_FUNCTION = &H0
Private Const TIME_CALLBACK_EVENT_SET = &H10

Private Declare Function timeSetEvent Lib "winmm.dll" _
    (ByVal uDelay As Long, ByVal uResolution As Long, ByVal lpFunction As Long, _
    ByVal dwUser As Long, ByVal uFlags As Long) As Long
Private Declare Function timeKillEvent Lib "winmm.dll" _
    (ByVal uID As Long) As Long
Private Declare Function CreateEvent Lib "Kernel32" Alias "CreateEventA" _
    (lpEventAttributes As SECURITY_ATTRIBUTES, ByVal bManualReset As Long, _
    ByVal bInitialState As Long, ByVal lpName As String) As Long
Private Declare Function WaitForMultipleObjects Lib "Kernel32" _
    (ByVal nCount As Long, lpHandles As Long, ByVal bWaitAll As Long, _
    ByVal dwMilliseconds As Long) As Long
Private Declare Function ResetEvent Lib "Kernel32" (ByVal hEvent As Long) As Long
Private Declare Function CloseHandle Lib "Kernel32" ( _
    ByVal hObject As Long) As Long

Private Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Long
End Type
    
```

## Classe pour attente avec timer multimedia

```

' Constante pour API d'attente
Private Const WAIT_ABANDONED& = &H80&
Private Const WAIT_ABANDONED_0& = &H80&
Private Const WAIT_FAILED& = -1&
Private Const WAIT_IO_COMPLETION& = &HC0&
Private Const WAIT_OBJECT_0& = 0
Private Const WAIT_OBJECT_1& = 1
Private Const WAIT_TIMEOUT& = &H102&
Private Const INFINITE = &HFFFF
Private Const QS_HOTKEY& = &H80
Private Const QS_KEY& = &H1
Private Const QS_MOUSEBUTTON& = &H4
Private Const QS_MOUSEMOVE& = &H2
Private Const QS_PAINT& = &H20
Private Const QS_POSTMESSAGE& = &H8
Private Const QS_SENDMESSAGE& = &H40
Private Const QS_TIMER& = &H10
Private Const QS_MOUSE& = (QS_MOUSEMOVE _
                          Or QS_MOUSEBUTTON)
Private Const QS_INPUT& = (QS_MOUSE _
                          Or QS_KEY)
Private Const QS_ALLEVENTS& = (QS_INPUT _
                              Or QS_POSTMESSAGE _
                              Or QS_TIMER _
                              Or QS_PAINT _
                              Or QS_HOTKEY)
Private Const QS_ALLINPUT& = (QS_SENDMESSAGE _
                              Or QS_PAINT _
                              Or QS_TIMER _
                              Or QS_POSTMESSAGE _
                              Or QS_MOUSEBUTTON _
                              Or QS_MOUSEMOVE _
                              Or QS_HOTKEY _
                              Or QS_KEY)

Private Declare Function MsgWaitForMultipleObjects Lib "User32" ( _
    ByVal nCount As Long, _
    pHandles As Long, _
    ByVal fWaitAll As Long, _
    ByVal dwMilliseconds As Long, _
    ByVal dwWakeMask As Long) As Long

Private gId As Long ' Id du timer
Private ghEvent As Long ' Id de l'événement
Private gInterval As Long ' Intervalle de l'événement

'-----
' Procédure d'attente
'-----

Public Sub Wait(ByVal pIntervalMs As Long)
    Dim lRet As Long
    ' On s'assure que l'intervalle est positif
    If pIntervalMs <= 1 Then pIntervalMs = 1
    ' Si non lancé ou si changement d'intervalle => on (re)lance le timer
    If pIntervalMs <> gInterval Then
        gInterval = pIntervalMs
        If gId <> 0 Then timeKillEvent gId
        gId = timeSetEvent(pIntervalMs, 0, ghEvent, 0, TIME_PERIODIC Or TIME_CALLBACK_EVENT_SET)
    End If
    ' Boucle tant que le timer n'a pas déclenché d'événement
    Do
        ' Cette fonction retourne si :
        '- le timer a été déclenché => lRet = WAIT_OBJECT_0
        '- n'importe quel autre message a été envoyé à l'application
        ' => on traite ces message avec un DoEvents
        lRet = MsgWaitForMultipleObjects(1, ghEvent, False, _
                                        INFINITE, QS_ALLINPUT&)

        ' Traite les messages
        DoEvents
    Loop Until lRet = WAIT_OBJECT_0

```

### Classe pour attente avec timer multimedia

```

End Sub

'-----
' Initialisation de la classe
'-----
Private Sub Class_Initialize()
    Dim ls As SECURITY_ATTRIBUTES
    ' Crée un événement
    ls.nLength = Len(ls)
    ls.lpSecurityDescriptor = 0
    ls.bInheritHandle = 0
    ghEvent = CreateEvent(ls, False, False, CStr(ObjPtr(Me)))
End Sub

'-----
' Fermeture de la classe
'-----
Private Sub Class_Terminate()
    ' Supprime les objets
    If gId <> 0 Then timeKillEvent gId
    If ghEvent <> 0 Then ResetEvent ghEvent
    If ghEvent <> 0 Then CloseHandle ghEvent
End Sub
    
```

Copier-coller ce code dans un nouveau module de classe que vous nommez **clWait**.

La fonction **Wait** permet d'attendre le déclenchement de la minuterie, tout en traitant les messages reçus par l'application.

Attention, si vous exécutez le code **Wait 10** dans une boucle, le programme n'attend pas 10 ms mais le déclenchement d'un timer toutes les 10 millisecondes.

On s'assure ainsi que le code qui suit l'instruction **Wait** est exécuté à intervalle régulier, quelque soit le temps d'exécution du reste du code.

(pour peu que le code s'exécute assez rapidement bien entendu).

La boucle principale devient :

### Boucle principale

```

'-----
' Boucle principale
'-----
Private Sub MainLoop()
    ' Création de la classe d'attente
    Dim lWait As clWait
    Set lWait = New clWait
    ' Flag pour sortie de la boucle
    gStop = False
    ' Début de la boucle
    Do
        lWait.Wait 1000 / 40 ' Attente et traitement des messages
        If gStop Then Exit Do ' Sortie de la boucle
        UpdateGame ' Mise à jour du jeu
        DisplayGame ' Affichage de l'image
    Loop
    ' Libération de la classe d'attente
    Set lWait = Nothing
End Sub
    
```

On attend 1000/40 millisecondes entre chaque affichage, ce qui nous donne un framerate de 40 images/seconde.

Vous pouvez exécuter le formulaire pour vous en assurer.

Faites ensuite varier ce temps d'attente; on peut définir le framerate avec précision.

Vous pouvez également vérifier le taux d'occupation du processeur (avec CTRL+ALT+SUPPR).

L'API **MsgWaitForMultipleObjects** a l'avantage de ne pas (ou peu) utiliser le CPU pendant l'attente.



## Module de déclaration pour les commandes

```

'-----
' Déclarations pour Joystick
'-----
Public Declare Function joyGetNumDevs Lib "winmm.dll" () As Long
Public Declare Function joyGetPos Lib "winmm.dll" _
    (ByVal uJoyID As Long, pji As JOYINFO) As Long
Public Declare Function joyGetDevCaps Lib "winmm.dll" Alias "joyGetDevCapsA" _
    (ByVal id As Long, lpCaps As JOYCAPS, ByVal uSize As Long) As Long
Public Declare Function joyGetPosEx Lib "winmm.dll" _
    (ByVal uJoyID As Long, pji As JOYINFOEX) As Long
Public Type JOYCAPS
    wMid As Integer
    wPid As Integer
    szPname As String * 32
    wXmin As Long
    wXmax As Long
    wYmin As Long
    wYmax As Long
    wZmin As Long
    wZmax As Long
    wNumButtons As Long
    wPeriodMin As Long
    wPeriodMax As Long
End Type
Public Type JOYINFOEX
    dwSize As Long
    dwFlags As Long
    dwXpos As Long
    dwYpos As Long
    dwZpos As Long
    dwRpos As Long
    dwUpos As Long
    dwVpos As Long
    dwButtons As Long
    dwButtonNumber As Long
    dwPOV As Long
    dwReserved1 As Long
    dwReserved2 As Long
End Type
Public Type JOYINFO
    X As Long
    Y As Long
    Z As Long
    Buttons As Long
End Type
Public Const JOY_BUTTON1 = &H1
Public Const JOY_BUTTON2 = &H2
Public Const JOY_BUTTON3 = &H4
Public Const JOY_BUTTON4 = &H8
Public Const JOYERR_BASE = 160
Public Const JOYERR_NOERROR = (0)
Public Const JOYERR_NOCANDO = (JOYERR_BASE + 6)
Public Const JOYERR_PARMS = (JOYERR_BASE + 5)
Public Const JOYERR_UNPLUGGED = (JOYERR_BASE + 7)
Public Const JOY_RETURNX As Long = &H1&
Public Const JOY_RETURNY As Long = &H2&
Public Const JOY_RETURNZ As Long = &H4&
Public Const JOY_RETURNR As Long = &H8&
Public Const JOY_RETURNU As Long = &H10
Public Const JOY_RETURNV As Long = &H20
Public Const JOY_RETURNPOV As Long = &H40&
Public Const JOY_RETURNBUTTONS As Long = &H80&
Public Const JOY_RETURNCENTERED As Long = &H400&
Public Const JOY_RETURNALL As Long = (JOY_RETURNX Or _
    JOY_RETURNY Or JOY_RETURNZ Or JOY_RETURNR Or _
    JOY_RETURNU Or JOY_RETURNV Or JOY_RETURNPOV Or JOY_RETURNBUTTONS)
'-----
    
```

### Module de déclaration pour les commandes

```
' Déclarations pour clavier
' http://msdn2.microsoft.com/en-us/library/ms646293(VS.85).aspx
-----
Public Declare Function GetAsyncKeyState Lib "User32" (ByVal vKey As Long) As Integer
Public Const VK_LEFT = &H25
Public Const VK_UP = &H26
Public Const VK_RIGHT = &H27
Public Const VK_DOWN = &H28
Public Const VK_SPACE = &H20
```

## III-A - Détection du clavier

La fonction **GetAsyncKeyState** nous donne l'état d'une touche du clavier.

Les constantes pour chaque touche sont données sur **MSDN**.

La fonction renvoie un résultat composé de deux entiers dans une variable de type long.

Un seul de ces deux entiers nous intéresse, d'où une opération logique (And &H8000) pour l'extraire.

### Procédure de gestion des commandes

```
'-----
' Procédure de gestion des commandes
'-----
Private Sub GestionCommandes()
' Etat des touches
    gKeyUp = (GetAsyncKeyState(VK_UP) And &H8000)
    gKeyDown = (GetAsyncKeyState(VK_DOWN) And &H8000)
    gKeyLeft = (GetAsyncKeyState(VK_LEFT) And &H8000)
    gKeyRight = (GetAsyncKeyState(VK_RIGHT) And &H8000)
    gKeySpace = (GetAsyncKeyState(VK_SPACE) And &H8000)
End Sub
```

## III-B - Détection du joystick

L'état du joystick sera déterminé grâce aux **API multimédia**.



**Consultez cet article de gRRosminet au sujet du contrôle du joystick sous windows**

Pour ce tutoriel, on se limitera à la détection du joystick d'indice 0.

Les axes utilisés pour le déplacement du vaisseau sont les axes X et Y.

Le bouton utilisé pour tirer est le bouton n° 2 (32 boutons possibles).

Il est bien entendu plus intéressant de proposer dans le jeu un choix du joystick et des boutons utilisés.

### Procédure de gestion des commandes

```
'-----
' Procédure de gestion des commandes
'-----
Private Sub GestionCommandes()
' Structure pour lecture de l'état du joystick
Dim linfo As JOYINFOEX
' Variable pour test si joystick présent
Dim lJoyPresent As Boolean
' Il est nécessaire d'initialiser cette variable
linfo.dwSize = Len(linfo)
' On lit l'état des axes X et Y, ainsi que des boutons
linfo.dwFlags = JOY_RETURNX Or JOY_RETURNY Or JOY_RETURNBUTTONS
' Lecture des infos et flag si joystick présent
lJoyPresent = (joyGetPosEx(0, linfo) = JOYERR_NOERROR)
' Etat des touches et du joystick
gKeyUp = (lJoyPresent And linfo.dwYpos = 0) Or _
(GetAsyncKeyState(VK_UP) And &H8000)
```

### Procédure de gestion des commandes

```

gKeyDown = (lJoyPresent And linfo.dwYpos = 65535) Or _
    (GetAsyncKeyState(VK_DOWN) And &H8000)
gKeyLeft = (lJoyPresent And linfo.dwXpos = 0) Or _
    (GetAsyncKeyState(VK_LEFT) And &H8000)
gKeyRight = (lJoyPresent And linfo.dwXpos = 65535) Or _
    (GetAsyncKeyState(VK_RIGHT) And &H8000)
gKeySpace = (lJoyPresent And (linfo.dwButtons And 4) = 4) Or _
    (GetAsyncKeyState(VK_SPACE) And &H8000)
    
```

End Sub

On limite la lecture à ce dont on a besoin avec les flags JOY\_RETURNX, JOY\_RETURN Y et JOY\_RETURNBUTTONS.

La fonction **joyGetPosEx** lit l'état du joystick d'indice 0 et renvoie JOYERR\_NOERROR si le joystick est présent.

On n'utilise pas les sticks analogiques pour les axes, uniquement les touches fléchées du joystick.

Leur état varie de 0 à 65535, 32767 étant la position intermédiaire.

L'état de tous les boutons est inscrit dans la variable **dwButtons**.

Si le bouton 4 est appuyé, cette variable vaut :  $2^4 = 16$ .

Si le bouton 6 est appuyé également, cette variable vaut :  $2^4 + 2^6 = 80$ .

Pour extraire l'état de chaque bouton, il faut faire une opération logique.

Le code permet maintenant de déplacer le vaisseau soit avec les touches fléchées, soit avec la croix directionnelle d'un joystick.

Pour tirer, il faut appuyer soit sur espace, soit sur le bouton 2 du joystick.

## IV - Gestion du son

Un jeu sans effet sonore est bien fade.

Pour ajouter du son à notre jeu, nous allons utiliser (encore une fois) une API multimedia : **mciSendString**.

Cette API nous permet notamment de jouer plusieurs sons simultanément.

Copiez-collez le code suivant dans un module nommé **ModSons** par exemple.

### Module pour gestion du son

```

'*****
'*                               MODULE POUR SON                               *
'*****
Option Explicit

Private Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" _
    (ByVal lpstrCommand As String, ByVal lpstrReturnString As String, _
    ByVal uReturnLength As Long, ByVal hwndCallback As Long) As Long

Private Declare Function GetShortPathName Lib "Kernel32" Alias "GetShortPathNameA" _
    (ByVal lpszLongPath As String, ByVal lpszShortPath As String, _
    ByVal cchBuffer As Long) As Long

Public Sub PlaySound(pPath As String)
Dim lPath As String
Dim lSize As Long
' Récupère le chemin court
lPath = pPath
lPath = Left(lPath, GetShortPathName(pPath, lPath, Len(pPath)))
' Stoppe le son si déjà en cours
mciSendString "Stop " & lPath, vbNullString, 0&, 0&
' Joue le son
mciSendString "Play " & lPath, vbNullString, 0&, 0&
End Sub

Public Sub StopSound(pPath As String)
Dim lPath As String
Dim lSize As Long
' Récupère le chemin court
lPath = pPath
lPath = Left(lPath, GetShortPathName(pPath, lPath, Len(pPath)))
    
```

### Module pour gestion du son

```
' Stoppe le son
mciSendString "Stop " & lPath, vbNullString, 0&, 0&
End Sub
```

Ce module ajoute deux fonctions :

- **PlaySound** pour jouer un son (wav ou midi).
- **StopSound** pour stopper un son.

Remarque : ce module est très simpliste, pour plus d'information sur cette API, [visitez le site MSDN](#)

Pour ajouter par exemple un son lors de l'envoi de missile :

### Ajout de son à la création de missiles dans la procédure UpdateGame

```
' Nouveaux missiles
If gKeySpace Then ' Si espace appuyé
    If Timer - sLastMissile > 0.2 Then ' Si pas de missile depuis plus de 0.2 secondes
        ' Joue un son
        PlaySound CurrentProject.Path & "\\fx\\LASERTW.WAV"
    [...]
End If
```

De nombreux sons peuvent être trouvés sur le net.

Beaucoup sont gratuits, certains pour une utilisation personnelle.

Pensez à vérifier la licence avant d'utiliser un son dans votre jeu.

Les sons de ce tutoriel ont été téléchargés sur : <http://www.freesoundfiles.tintagel.net/Audio/>

## V - Conclusion

On a progressé dans ce tutoriel :

- la vitesse du jeu est maîtrisée (au moins sur les ordinateurs assez rapide)
- on peut déplacer le vaisseau au joystick.
- on a ajouté du son, ce qui rend le jeu beaucoup plus "vivant"



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

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

## VI - Téléchargements

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

**Télécharger le classeur Excel de ce tutoriel au format EXCEL 2000 (HTTP)**