Université Pierre et Marie Curie
DEUST
Année 2000–2001

Module de programmation

Projet Space Invader

Bon, tout le monde sait ce qu'est un Space Invader. Donc, je ne vais pas donner d'explications détaillées sur le jeu lui-même. Par contre, vous trouverez ci-dessous quelques informations utiles pour la réalisation du projet.

1 Indications sur la mise en oeuvre du projet

1.1 Création du projet sous Visual C++

Le Space Invader est un jeu graphique. Pour créer votre projet sous Visual C++, vous allez donc sélectionner une "win32 application". Visual vous demandera alors quel type d'application vous voulez ouvrir ("empty projet", "simple application", etc). Sélectionnez "typical hello world". Cela créera la fonction `main()` (qui s'appelle ici `WinMain()`).

Vous n'avez pas besoin de regarder les fonctions créées par Visual sauf celle dont le nom est : `WndProc()`. Cette fonction récupère tous les événements gérés par l'environnement graphique (déplacement de souris, tape d'une touche au clavier, etc). Les événements qui vous intéressent sont les suivants :

événement	cause
WM_CREATE	quand on crée la fenêtre
WM_COMMAND	quand on sélectionne un item dans un menu
WM_PAINT	repeind la fenêtre (lors des créations/agrandissements)
WM_DESTROY	quand on détruit la fenêtre
WM_TIMER	lorsqu'un timer se réveille
WM_KEYDOWN	quand l'utilisateur appuie sur une touche

Votre fonction devra récupérer ces événements afin de savoir ce que fait l'utilisateur. À noter que lorsqu'un WM_TIMER arrive, le paramètre `wParam` de la fonction `WndProc()` contient le numéro d'identification du timer. Et lorsqu'un WM_KEYDOWN est récupéré par `WndProc()`, alors `(char) wParam` vaut le code ascii du caractère tapé par le joueur.

1.2 Utilisation des fonctions graphiques

Afin d'obtenir un jeu un tant soit peu rapide, vous ne dessinerez pas les vaisseaux spatiaux et les missiles grâce aux fonctions permettant de tracer des lignes, des rectangles ou des ellipses, mais vous utiliserez plutôt des bitmaps. L'idée est de sauvegarder les images des vaisseaux et des missiles dans des bitmaps. Lors du démarage de l'application, on charge ces bitmaps, appelons-les $B_1, \ldots, B_n$. On crée en outre un nouveau bitmap de la taille de la feneêtre, appelons-le X. Quand on veut raffraichir l'affichage de la fenêtre, on commence par rendre le bitmap X tout blanc (en utilisant la fonction `BibBlt` avec l'option `WHITENESS` (cf. l'annexe de ce poly). Ensuite, on copie chacun des bitmaps B_i aux endroits que l'on veut dans X, là encore en utilisant la fonction `BibBlt`. Enfin on copie X à l'écran en utilisant une dernière fois la fonction `BibBlt`. Il n'y a pas de scintillement à l'écran car tous les calculs ont été réalisés dans X et non à l'écran, et c'est seulement lorsque X a été correctement mis à jour que l'on a raffraichi l'écran.

La création des bitmaps s'effectue dans les ressources : sur la partie gauche de l'éditeur de Visual C++, vous avez un "panel" qui vous indique la liste des fonctions et des variables globales de votre projet. En bas, vous avez trois onglets : "classes", "ressources", "files". Cliquez sur l'onglet "ressources". Vous verrez alors apparaître la liste de toutes les ressources affectées à votre projet (icones, raccourcis clavier, menus, etc). Sélectionnez maintenant dans le menu `projet` la case `add ressources` et cliquez sur `bitmap`. Vous venez de créer un nouveau bitmap. Dans la

partie droite de la fenêtre, vous pouvez dessinez ce que vous voulez dans le bitmap. Celui-ci sera sauvegardé automatiquement par Visual C++ lorsque vous quitterez l'éditeur ou bien lorsque vous sauvegarderez le workspace.

Afin de vous familiariser avec ces fonctions, vous étudierez les programmes suivants, qui se trouvent dans le répertoire `L:\vc6` :

Nom du fichier	Description
ChargeBitmap	Ce projet montre comment on peut charger une image en mémoire au démarrage de l'exécutable. La partie intéressante à lire se trouve dans le `case WM_CREATE` de la fonction `WndProc()`.
CreeBitmap	Ce projet montre comment on peut créer un bitmap ayant exactement la taille de la fenêtre. La partie intéressante à lire se trouve dans le `case WM_PAINT` de la fonction `WndProc()`.
AfficheBitmap	Ce projet illustre la manière d'afficher un bitmap à l'écran. La fonction intéressante à lire s'appelle `affichage()`.
AfficheTexte	ce projet montre comment afficher un texte à l'écran. La partie intéressante se trouve dans le `case WM_PAINT` de la fonction `WndProc()`.
LitUneTouche	Ce projet illustre la manière de récupérer une touche entrée au clavier par l'utilisateur. Cette touche est alors affichée à l'écran. La partie intéressante à lire se trouve dans le `case WM_KEYDOWN` de la fonction `WndProc()`.

1.3 un squelette de programme

Afin de vous aider à démarrer ce projet, voilà une liste de fonctions que je vous propose d'écrire. Vous n'êtes pas tenu de le faire. Si vous préférez découper votre programme autrement, libre à vous du moment que ç a marche à la fin.

Nom de la fonction	Description
cree_bitmaps	Comme son nom l'indique, cette fonction charge les bitmaps des vaisseaux spatiaux et des missiles en mémoire.
affiche	cette fonction raffraichit l'écran, c'est-à-dire qu'elle redessine tous les vaisseaux spatiaux, ainsi que les missiles et le score du joueur et le nombre de vies restantes.
deplace_ennemi	cette fonction réalise le déplacement des ennemis non encore détruis.
init_jeu	cette fonction initialise les paramètres pour commencer une nouvelle partie.
tir_ennemi	cette fonction fait tirer certains des vaisseaux ennemis les plus bas sur l'écran.
deplace_missiles	cette fonction met à jour la position des missiles, c'est-à-dire qu'elle déplace les missiles du joueur d'un cran vers le haut et ceux des ennemis d'un cran vers le bas. De plus, elle élimine les missiles qui ne sont plus dans l'espace de jeu.
ennemi_action	en fonction de timers, fait déplacer les ennemis ou bien les fait tirer.
action_joueur	réalise les actions demandées par le joueur en fonction de la touche appuyée par ce dernier.
explose_missiles	réalise l'explosion des missiles en contact avec les ennemis ou avec le joueur.

2 Travail à rendre

Vous me rendrez sur une disquette votre projet bien évidemment. Pour que cela tienne sur la disquette, vous détruirez le sous-répertoire "debug" de votre projet (il est gros et peut être regénéré sans problème). Vous porterez une attention particulière à la mise en page de votre programme (mettez des commentaire et séparez bien les fonctions).

Vous rendrez en outre un petit rapport expliquant la structure de vos données ainsi que quelques explications sur les fonctions que vous aurez écrites.

3 Annexe : quelques fonctions graphiques utiles

Dans cette section, vous trouverez les explications données à propos des fonctions graphiques suivantes par la MSDN library :

- BitBlt()
- CreateCompatibleDC()

- DeleteDC()
- GetDC()
- GetObject()
- LoadImage()
- ReleaseDC()
- SelectObject()
- SetTimer()

3.1 BitBlt

The BitBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

```
BOOL BitBlt(
  HDC hdcDest, // handle to destination device context
  int nXDest,  // x-coordinate of destination rectangle's upper-left
               // corner
  int nYDest,  // y-coordinate of destination rectangle's upper-left
               // corner
  int nWidth,  // width of destination rectangle
  int nHeight, // height of destination rectangle
  HDC hdcSrc,  // handle to source device context
  int nXSrc,   // x-coordinate of source rectangle's upper-left
               // corner
  int nYSrc,   // y-coordinate of source rectangle's upper-left
               // corner
  DWORD dwRop  // raster operation code
);
```

Parameters :

<table>
<tr><td>hdcDest</td><td colspan="2">Handle to the destination device context.</td></tr>
<tr><td>nXDest</td><td colspan="2">Specifies the logical x-coordinate of the upper-left corner of the destination rectangle.</td></tr>
<tr><td>nYDest</td><td colspan="2">Specifies the logical y-coordinate of the upper-left corner of the destination rectangle.</td></tr>
<tr><td>nWidth</td><td colspan="2">Specifies the logical width of the source and destination rectangles.</td></tr>
<tr><td>nHeight</td><td colspan="2">Specifies the logical height of the source and the destination rectangles.</td></tr>
<tr><td>hdcSrc</td><td colspan="2">Handle to the source device context.</td></tr>
<tr><td>nXSrc</td><td colspan="2">Specifies the logical x-coordinate of the upper-left corner of the source rectangle.</td></tr>
<tr><td>nYSrc</td><td colspan="2">Specifies the logical y-coordinate of the upper-left corner of the source rectangle.</td></tr>
<tr><td>dwRop</td><td colspan="2">Specifies a raster-operation code. These codes define how the color data for the source rectangle is to be combined with the color data for the destination rectangle to achieve the final color. The following list shows some common raster operation codes.
<table>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>BLACKNESS</td><td>Fills the destination rectangle using the color associated with index 0 in the physical palette. (This color is black for the default physical palette.)</td></tr>
<tr><td>DSTINVERT</td><td>Inverts the destination rectangle.</td></tr>
<tr><td>MERGECOPY</td><td>Merges the colors of the source rectangle with the specified pattern by using the Boolean AND operator.</td></tr>
<tr><td>MERGEPAINT</td><td>Merges the colors of the inverted source rectangle with the colors of the destination rectangle by using the Boolean OR operator.</td></tr>
<tr><td>NOMIRRORBITMAP</td><td>Windows NT 5.0 and later, Windows 98: Prevents the bitmap from being mirrored</td></tr>
<tr><td>NOTSRCCOPY</td><td>Copies the inverted source rectangle to the destination.</td></tr>
<tr><td>NOTSRCERASE</td><td>Combines the colors of the source and destination rectangles by using the Boolean OR operator and then inverts the resultant color.</td></tr>
<tr><td>PATCOPY</td><td>Copies the specified pattern into the destination bitmap.</td></tr>
<tr><td>PATINVERT</td><td>Combines the colors of the specified pattern with the colors of the destination rectangle by using the Boolean XOR operator.</td></tr>
<tr><td>PATPAINT</td><td>Combines the colors of the pattern with the colors of the inverted source rectangle by using the Boolean OR operator. The result of this operation is combined with the colors of the destination rectangle by using the Boolean OR operator.</td></tr>
<tr><td>SRCAND</td><td>Combines the colors of the source and destination rectangles by using the Boolean AND operator.</td></tr>
<tr><td>SRCCOPY</td><td>Copies the source rectangle directly to the destination rectangle.</td></tr>
<tr><td>SRCERASE</td><td>Combines the inverted colors of the destination rectangle with the colors of the source rectangle by using the Boolean AND operator.</td></tr>
<tr><td>SRCINVERT</td><td>Combines the colors of the source and destination rectangles by using the Boolean XOR operator.</td></tr>
<tr><td>SRCPAINT</td><td>Combines the colors of the source and destination rectangles by using the Boolean OR operator.</td></tr>
<tr><td>WHITENESS</td><td>Fills the destination rectangle using the color associated with index 1 in the physical palette. (This color is white for the default physical palette.)</td></tr>
</table></td></tr>
</table>

Return Values :

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero.
Windows NT: To get extended error information, call GetLastError.

Remarks :

If a rotation or shear transformation is in effect in the source device context, BitBlt returns an error. If other transformations exist in the source device context (and a matching transformation

is not in effect in the destination device context), the rectangle in the destination device context is stretched, compressed, or rotated, as necessary.
If the color formats of the source and destination device contexts do not match, the BitBlt function converts the source color format to match the destination format.
When an enhanced metafile is being recorded, an error occurs if the source device context identifies an enhanced-metafile device context.
Not all devices support the BitBlt function. For more information, see the RC_BITBLT raster capability entry in the GetDeviceCaps function as well as the following functions: MaskBlt, PlgBlt, and StretchBlt.
BitBlt returns an error if the source and destination device contexts represent different devices.

3.2 CreateCompatibleDC

The CreateCompatibleDC function creates a memory device context (DC) compatible with the specified device.

```
HDC CreateCompatibleDC(
  HDC hdc   // handle to the device context
);
```

Parameters :

hdc	Handle to an existing device context. If this handle is NULL, the function creates a memory device context compatible with the application's current screen.

Return Values :

If the function succeeds, the return value is the handle to a memory device context.
If the function fails, the return value is NULL.
Windows NT: To get extended error information, call GetLastError.

Remarks :

A memory device context is a device context that exists only in memory. When the memory device context is created, its display surface is exactly one monochrome pixel wide and one monochrome pixel high. Before an application can use a memory device context for drawing operations, it must select a bitmap of the correct width and height into the device context. This may be done by using CreateCompatibleBitmap specifying the height, width, and color organization required in the function call.
When a memory device context is created, all attributes are set to normal default values. The memory device context can be used as a normal device context. You can set the attributes to nondefault values, obtain the current setting of its attributes, select pens, brushes, and regions into it.
The CreateCompatibleDC function can only be used with devices that support raster operations. An application can determine whether a device supports these operations by calling the GetDeviceCaps function.
When you no longer need the memory device context, call the DeleteDC function to delete it.

3.3 DeleteDC

The DeleteDC function deletes the specified device context (DC).

```
BOOL DeleteDC(
  HDC hdc   // handle to device context
);
```

Parameters :

hdc	Handle to the device context.

Return Values :

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero.
Windows NT: To get extended error information, call GetLastError.

Remarks :

An application must not delete a device context whose handle was obtained by calling the GetDC function. Instead, it must call the ReleaseDC function to free the device context.

3.4 GetDC

The GetDC function retrieves a handle to a display device context for the client area of a specified window or for the entire screen. You can use the returned handle in subsequent GDI functions to draw in the device context.

The GetDCEx function is an extension to GetDC, which gives an application more control over how and whether clipping occurs in the client area.

```
HDC GetDC(
  HWND hWnd   // handle to a window
);
```

Parameters :

hWnd	Handle to the window whose device context is to be retrieved. If this value is NULL, GetDC retrieves the device context for the entire screen. Windows 98, Windows NT 5.0 and later: If this parameter is NULL, GetDC retrieves the device context for the primary display monitor. To get the device context for other display monitors, use the EnumDisplayMonitors and CreateDC functions.

Return Values :

If the function succeeds, the return value identifies the device context for the specified window's client area.
If the function fails, the return value is NULL.
Windows NT: To get extended error information, call GetLastError.

Remarks :

The GetDC function retrieves a common, class, or private device context depending on the class style specified for the specified window. For common device contexts, GetDC assigns default attributes to the device context each time it is retrieved. For class and private device contexts, GetDC leaves the previously assigned attributes unchanged.
After painting with a common device context, the ReleaseDC function must be called to release the device context. Class and private device contexts do not have to be released. The number of device contexts is limited only by available memory.

3.5 GetObject

The GetObject function obtains information about a specified graphics object. Depending on the graphics object, the function places a filled-in BITMAP, DIBSECTION, EXTLOGPEN, LOGBRUSH, LOGFONT, or LOGPEN structure, or a count of table entries (for a logical palette), into a specified buffer.

```
int GetObject(
  HGDIOBJ hgdiobj,  // handle to graphics object of interest
  int cbBuffer,     // size of buffer for object information
  LPVOID lpvObject  // pointer to buffer for object information
);
```

Parameters :

<table>
<tr><td>hgdiobj</td><td>Handle to the graphics object of interest. This can be a handle to one of the following: a logical bitmap, a brush, a font, a palette, a pen, or a device independent bitmap created by calling the CreateDIBSection function.</td></tr>
<tr><td>cbBuffer</td><td>Specifies the number of bytes of information to be written to the buffer.</td></tr>
<tr><td>lpvObject</td><td>Pointer to a buffer that is to receive the information about the specified graphics object. The following table shows the type of information the buffer receives for each type of graphics object you can specify with hgdiobj:
<table>
<tr><td>Object</td><td>type</td><td>Data written to *lpvObject</td></tr>
<tr><td>HBITMAP</td><td>BITMAP</td><td>HBITMAP returned from a call to CreateDIBSection DIBSECTION, if cbBuffer is set to sizeof(DIBSECTION), or BITMAP, if cbBuffer is set to sizeof(BITMAP)</td></tr>
<tr><td>HPALETTE</td><td>A WORD</td><td>count of the number of entries in the logical palette</td></tr>
<tr><td>HPEN</td><td>LOGPEN</td><td>returned from a call to ExtCreatePen EXTLOGPEN</td></tr>
<tr><td>HBRUSH</td><td>LOGBRUSH</td><td></td></tr>
<tr><td>HFONT</td><td>LOGFONT</td><td></td></tr>
<tr><td colspan="3">If the lpvObject parameter is NULL, the function return value is the number of bytes required to store the information it writes to the buffer for the specified graphics object.</td></tr>
</table></td></tr>
</table>

Return Values :

If the function succeeds, and lpvObject is a valid pointer, the return value is the number of bytes stored into the buffer.
If the function succeeds, and lpvObject is NULL, the return value is the number of bytes required to hold the information the function would store into the buffer.
If the function fails, the return value is zero.
Windows NT: To get extended error information, call GetLastError.

Remarks :

The buffer pointed to by the lpvObject parameter must be sufficiently large to receive the information about the graphics object.
If hgdiobj identifies a bitmap created by calling CreateDIBSection, and the specified buffer is large enough, the GetObject function returns a DIBSECTION structure. In addition, the bmBits member of the BITMAP structure contained within the DIBSECTION will contain a pointer to the bitmap's bit values.
If hgdiobj identifies a bitmap created by any other means, GetObject returns only the width, height, and color format information of the bitmap. You can obtain the bitmap's bit values by calling the GetDIBits or GetBitmapBits function.
If hgdiobj identifies a logical palette, GetObject retrieves a 2-byte integer that specifies the number of entries in the palette. The function does not retrieve the LOGPALETTE structure defining the palette. To retrieve information about palette entries, an application can call the GetPaletteEntries function.

3.6 LoadImage

The LoadImage function loads an icon, cursor, or bitmap.

```
HANDLE LoadImage(
  HINSTANCE hinst,   // handle of the instance containing the image
  LPCTSTR lpszName,  // name or identifier of image
  UINT uType,        // type of image
  int cxDesired,     // desired width
  int cyDesired,     // desired height
  UINT fuLoad        // load flags
);
```

Parameters :

<table>
<tr><td>hinst</td><td>Handle to an instance of the module that contains the image to be loaded. To load an OEM image, set this parameter to zero.</td></tr>
<tr><td>lpszName</td><td>Identifies the image to load. If the hinst parameter is non-NULL and the fuLoad parameter omits LR_LOADFROMFILE, lpszName specifies the image resource in the hinst module. If the image resource is to be loaded by name, the lpszName parameter is a pointer to a null-terminated string that contains the name of the image resource. If the image resource is to be loaded by ordinal, use the MAKEINTRESOURCE macro to convert the image ordinal into a form that can be passed to the LoadImage function.
If the hinst parameter is NULL and the fuLoad parameter omits the LR_LOADFROMFILE value, the lpszName specifies the OEM image to load. The OEM image identifiers are defined in Winuser.h and have the following prefixes.
<table>
<tr><th>Prefix</th><th>Meaning</th></tr>
<tr><td>OBM_</td><td>OEM bitmaps</td></tr>
<tr><td>OIC_</td><td>OEM icons</td></tr>
<tr><td>OCR_</td><td>OEM cursors</td></tr>
</table>
To pass these constants to the LoadImage function, use the MAKEINTRESOURCE macro. For example, to load the OCR_NORMAL cursor, pass MAKEINTRESOURCE(OCR_NORMAL) as the lpszName parameter and NULL as the hinst parameter.
If the fuLoad parameter includes the LR_LOADFROMFILE value, lpszName is the name of the file that contains the image.</td></tr>
<tr><td>uType</td><td>Specifies the type of image to be loaded. This parameter can be one of the following values.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>IMAGE_BITMAP</td><td>Loads a bitmap.</td></tr>
<tr><td>IMAGE_CURSOR</td><td>Loads a cursor.</td></tr>
<tr><td>IMAGE_ICON</td><td>Loads an icon.</td></tr>
</table></td></tr>
<tr><td>cxDesired</td><td>Specifies the width, in pixels, of the icon or cursor. If this parameter is zero and the fuLoad parameter is LR_DEFAULTSIZE, the function uses the SM_CXICON or SM_CXCURSOR system metric value to set the width. If this parameter is zero and LR_DEFAULTSIZE is not used, the function uses the actual resource width.</td></tr>
<tr><td>cyDesired</td><td>Specifies the height, in pixels, of the icon or cursor. If this parameter is zero and the fuLoad parameter is LR_DEFAULTSIZE, the function uses the SM_CYICON or SM_CYCURSOR system metric value to set the height. If this parameter is zero and LR_DEFAULTSIZE is not used, the function uses the actual resource height.</td></tr>
<tr><td>fuLoad</td><td>Specifies a combination of the following values.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>LR_DEFAULTCOLOR</td><td>The default flag; it does nothing. All it means is ”not LR_MONOCHROME”.</td></tr>
<tr><td>LR_CREATEDIBSECTION</td><td>When the uType parameter specifies IMAGE_BITMAP, causes the function to return a DIB section bitmap rather than a compatible bitmap. This flag is useful for loading a bitmap without mapping it to the colors of the display device.</td></tr>
<tr><td>LR_DEFAULTSIZE</td><td>Uses the width or height specified by the system metric values for cursors or icons, if the cxDesired or cyDesired values are set to zero. If this flag is not specified and cxDesired and cyDesired are set to zero, the function uses the actual resource size. If the resource contains multiple images, the function uses the size of the first image.</td></tr>
<tr><td>LR_LOADFROMFILE</td><td>Loads the image from the file specified by the lpszName parameter. If this flag is not specified, lpszName is the name of the resource.</td></tr>
<tr><td>LR_LOADTRANSPARENT</td><td>Retrieves the color value of the first pixel in the image and replaces the corresponding entry in the color table with the default window color (COLOR_WINDOW). All pixels in the image that use that entry become the default window color. This value applies only to images that have corresponding color tables. Do not use this option if you are loading a bitmap with a color depth greater than 8bpp.</td></tr>
<tr><td>LR_MONOCHROME</td><td>Loads the image in black and white.</td></tr>
</table></td></tr>
</table>

Return Values :

If the function succeeds, the return value is the handle of the newly loaded image.
If the function fails, the return value is NULL. To get extended error information, call GetLastError.

Remarks :

When you are finished using a bitmap, cursor, or icon you loaded without specifying the LR_SHARED flag, you can release its associated memory by calling one of the functions in the following table.

Resource	Release function
Bitmap	DeleteObject
Cursor	DestroyCursor
Icon	DestroyIcon

The system automatically deletes these resources when the process that created them terminates, however, calling the appropriate function saves memory and decreases the size of the process's working set.
If you are targeting a platform that does not support mouse cursors, you cannot specify the SM_CXCURSOR and SM_CYCURSOR values in the cxDesired and cyDesired parameters, and you cannot specify IMAGE_CURSOR for the uType parameter.
If you are targeting a platform that supports mouse cursors, you can specify SM_CXCURSOR and SM_CYCURSOR in the cxDesired and cyDesired parameters, and IMAGE_CURSOR in the uType parameter.

3.7 ReleaseDC

The ReleaseDC function releases a device context (DC), freeing it for use by other applications. The effect of the ReleaseDC function depends on the type of device context. It frees only common and window device contexts. It has no effect on class or private device contexts.

```
int ReleaseDC(
  HWND hWnd,  // handle to window
  HDC hDC     // handle to device context
);
```

Parameters :

hWnd	Handle to the window whose device context is to be released.
hDC	Handle to the device context to be released.

Return Values :

The return value specifies whether the device context is released. If the device context is released, the return value is 1.
If the device context is not released, the return value is zero.

Remarks :

The application must call the ReleaseDC function for each call to the GetWindowDC function and for each call to the GetDC function that retrieves a common device context.
An application cannot use the ReleaseDC function to release a device context that was created by calling the CreateDC function; instead, it must use the DeleteDC function.

3.8 SelectObject

The SelectObject function selects an object into the specified device context. The new object replaces the previous object of the same type.

```
HGDIOBJ SelectObject(
  HDC hdc,          // handle to device context
  HGDIOBJ hgdiobj   // handle to object
);
```

Parameters :

hdc	Handle to the device context.	
hgdiobj	Handle to the object to be selected. The specified object must have been created by using one of the following functions.	
	Object	Functions
	Bitmap	CreateBitmap, CreateBitmapIndirect, CreateCompatibleBitmap, CreateDIBitmap, CreateDIBSection (Bitmaps can be selected for memory device contexts only, and for only one device context at a time.)
	Brush	CreateBrushIndirect, CreateDIBPatternBrush, CreateDIBPatternBrushPt, CreateHatchBrush, CreatePatternBrush, CreateSolidBrush Font CreateFont, CreateFontIndirect
	Pen	CreatePen, CreatePenIndirect
	Region	CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreateRectRgn, CreateRectRgnIndirect

Return Values :

If the selected object is not a region and the function succeeds, the return value is the handle of the object being replaced. If the selected object is a region and the function succeeds, the return value is one of the following values.

Value	Meaning
SIMPLEREGION	Region consists of a single rectangle.
COMPLEXREGION	Region consists of more than one rectangle.
NULLREGION	Region is empty.

If an error occurs and the selected object is not a region, the return value is NULL. Otherwise, it is GDI_ERROR.

Remarks :

This function returns the previously selected object of the specified type. An application should always replace a new object with the original, default object after it has finished drawing with the new object.
An application cannot select a bitmap into more than one device context at a time.

3.9 SetTimer

The SetTimer function creates a timer with the specified time-out value.

```
UINT SetTimer(
  HWND hWnd,              // handle to window for timer messages
  UINT nIDEvent,          // timer identifier
  UINT uElapse,           // time-out value
  TIMERPROC lpTimerFunc   // pointer to timer procedure
);
```

Parameters :

hWnd	Handle to the window to be associated with the timer. This window must be owned by the calling thread. If this parameter is NULL, no window is associated with the timer and the nIDEvent parameter is ignored.
nIDEvent	Specifies a nonzero timer identifier. If the hWnd parameter is NULL, this parameter is ignored. If the hWnd parameter is not NULL and the window specified by hWnd already has a timer with the value nIDEvent, then the existing timer is replaced by the new timer.
uElapse	Specifies the time-out value, in milliseconds.
lpTimerFunc	Pointer to the function to be notified when the time-out value elapses. For more information about the function, see TimerProc. If lpTimerFunc is NULL, the system posts a WM_TIMER message to the application queue. The hwnd member of the message's MSG structure contains the value of the hWnd parameter.

Return Values :

If the function succeeds, the return value is an integer identifying the new timer. An application can pass this value, or the string identifier, if it exists, to the KillTimer function to destroy the timer.
If the function fails to create a timer, the return value is zero. To get extended error information, call GetLastError.

Remarks :

An application can process WM_TIMER messages by including a WM_TIMER case statement in the window procedure or by specifying a TimerProc callback function when creating the timer. When you specify a TimerProc callback function, the default window procedure calls the callback function when it processes WM_TIMER. Therefore, you need to dispatch messages in the calling thread, even when you use TimerProc instead of processing WM_TIMER.
The wParam parameter of the WM_TIMER message contains the value of the nIDEvent parameter. The timer identifier, nIDEvent, is specific to the associated window. Another window can have its own timer which has the same identifier as a timer owned by another window. The timers are distinct.