Université Pierre et Marie Curie
DEUST
Année 2001–2002

Module de programmation
© C. Gonzales 2001

Projet Course de voitures

1 Description du projet

Comme le titre l'indique, le projet consiste à réaliser une course de voitures en 3D. Vous devrez alors obtenir une application ressemblant à l'image ci-dessous : En bas, au centre, de l'écran se trouve la voiture que vous

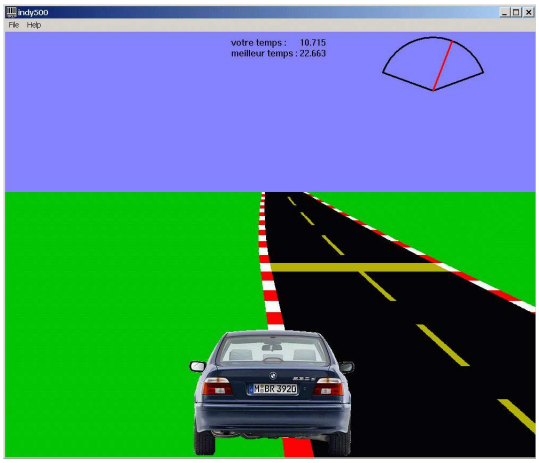


FIG. 1 – Une image du projet une fois réalisé.

conduisez. C'est en fait un bitmap qui vous est fourni. Plus exactement, cinq bitmaps vous sont fournis, qui représentent la voiture dans différentes positions : Vous pouvez accélérer et décélérer avec les flèches vers le

FIG. 2 – Les bitmaps des voitures.

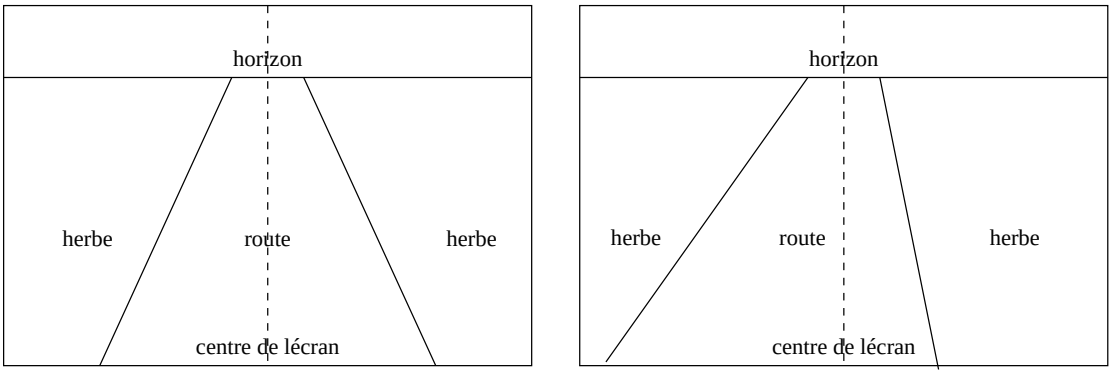
haut et vers le bas. Pour simplifier, le passage des vitesses se fait automatiquement. Ainsi, si vous décélérez suffisamment, vous repartez en arrière. Vous pouvez aussi incliner le volant à droite ou à gauche avec les flèches vers la droite et vers la gauche. En haut de l'écran, sur la droite, vous pouvez voir votre compteur de vitesse et, à sa gauche, le meilleur temps pour boucler le circuit ainsi que votre chrono actuel. Le programme vous fait tourner indéfiniment sur le même circuit. Optionnellement, vous pouvez créer un item dans un menu afin de changer de circuit.

2 L'algorithme de défilement de la piste

2.1 L'affichage grâce à des lignes horizontales

Bien entendu, la complexité du projet réside dans la manière de simuler le défilement de la route en 3D sur un écran 2D. Dans les jeux actuels, on se contente de décrire le paysage comme une série de triangles et c'est la carte graphique qui effectue les calculs pour placer ces triangles à l'écran et les remplir avec une texture. Dans le jeu qui nous concerne, nous allons plutôt revenir aux temps anciens, ceux des premiers PC. À cette époque, c'était au programmeur de décrire explicitement ce qui devait être affiché à l'écran (tracer une ligne entre tel et tel points, tracer un cercle ici, un rectangle là, etc).

Problème : comment tracer à la main une route en perspective ? Eh bien, si l'on utilise une petite astuce, c'est relativement simple. Considérons tout d'abord le cas d'une route rectiligne au centre de laquelle on se trouve. Le conducteur devrait voir un paysage similaire à la figure 3.a. De même, si l'on se trouve sur le côté d'une route toute droite, on devrait observer un paysage similaire à celui de la figure 3.b. On pourrait donc imaginer



a) si l'on est au centre de la route

b) si l'on est sur la droite de la route

FIG. 3 – Le cas d'une route rectiligne.

de tracer la route grâce à des lozanges. Cela fonctionnerait très bien tant que l'on aurait des routes droites mais, malheureusement, dès que l'on aurait un tournant, cela ne marcherait plus. Dans le cas d'un tournant, la route devrait avoir un aspect similaire à celui de la figure 4.

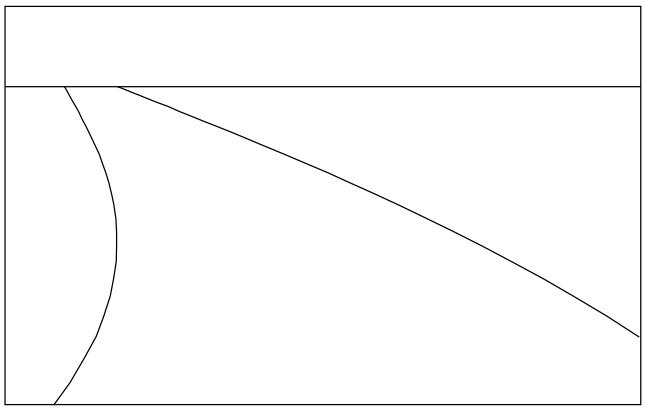


FIG. 4 – Le cas d'une route curviligne.

Il faudrait donc repenser l'algorithme d'affichage de la route de manière à ce qu'il fonctionne aussi bien dans le cas d'une route rectiligne que dans le cas d'une route non rectiligne. L'idée est alors relativement simple : pensons la route comme une succession de lignes horizontales. Les trois figures ci-dessus rentrent dans ce modèle, et c'est celui que nous allons adopter. Ainsi :

> La route est une succession de lignes horizontales. À une hauteur donnée correspond une longueur de ligne précise, indépendante de la courbure de la route. L'algorithme d'affichage consiste donc juste à calculer la position horizontale des bords gauches des lignes horizontales.

2.2 Le rendu de la perspective

Évidemment, pour donner une impression de perspective, les lignes horizontales seront de plus en plus petites au fur et à mesure que l'on remonte vers le haut de l'écran. Comme vous avez pu le constater sur la première page, la route comporte des bandes latérales rouges et blanches. Là encore, afin de donner un effet de perspective, les bandes doivent avoir une hauteur de plus en plus petite au fur et à mesure que l'on remonte vers le haut de l'écran. C'est pourquoi l'écran est divisé en un certain nombre de segments, ceux-ci étant de plus en plus petits vers le haut.

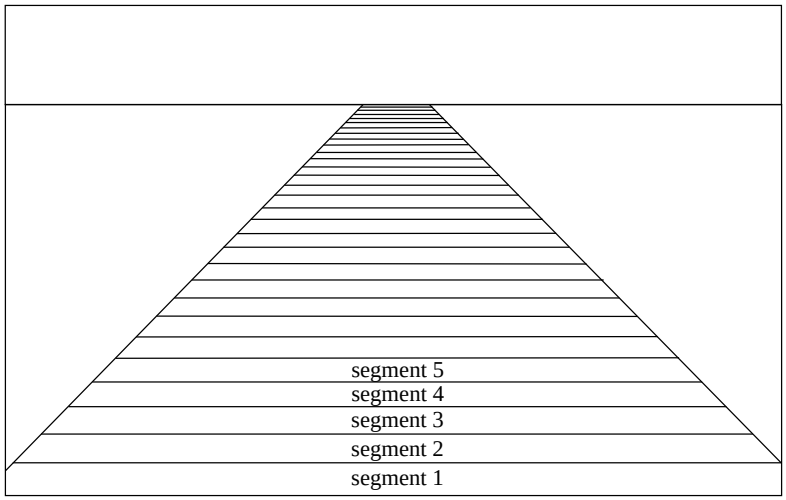


FIG. 5 – Les segments qui donnent un effet de perspective.

Pour déterminer leur taille, j'ai simplement appliqué le ratio suivant :

$$\frac{\text{hauteur segment numéro } i+1}{\text{hauteur segment numéro } i} = 0,93.$$

2.3 L'algorithme d'affichage proprement dit

L'algorithme consiste à séparer l'écran en un certain nombre de segments (sur mon portable, j'ai utilisé 30 segments, mais ma résolution graphique doit être plus élevée que celle des machines du DEUST). Lors de l'initialisation du programme, on définit une fois pour toutes où se trouvent chacun des segments (entre tel et tel pixel de hauteur). Au début de la procédure d'affichage, on détermine la position horizontale de la ligne la plus basse de l'écran (donc dans le segment numéro 1 sur la figure ci-dessus).

Supposons maintenant que la route soit rectiligne. Dans ce cas, la ligne horizontale la plus haute à afficher devrait se trouver au centre de l'écran. La fonction d'affichage calcule donc la position horizontale de cette ligne. Si la route était rectiligne, l'ensemble des bords gauches des lignes horizontales devrait former une droite, comme indiqué sur la figure 6. Ah oui mais alors le calcul de la position des bords gauches des lignes est très

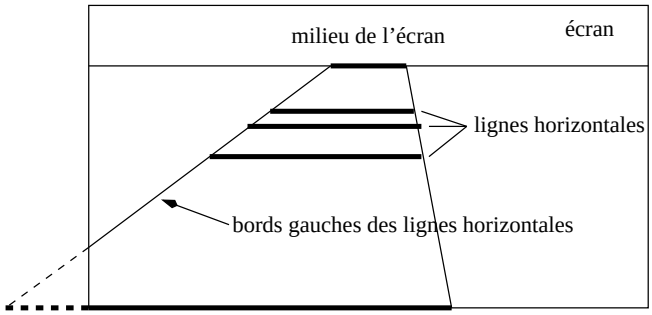


FIG. 6 – Calcul de la position des bords gauches.

simple puisque tout est linéaire : si (x_0, y_0) représente la position du bord gauche de la ligne la plus basse et (x_n, y_n) celle du bord gauche de la ligne la plus haute, et si (x, y) représente la position du bord gauche de la

$y^{\text{ème}}$ ligne, alors :

$$x = \frac{x_n - x_0}{y_n - y_0}(y - y_0).$$

On peut alors calculer quel est le déplacement d_x quand on se décale de d_y pixels vers le haut :

$$x + d_x = \frac{x_n - x_0}{y_n - y_0}(y + d_y - y_0) \Rightarrow d_x = \frac{x_n - x_0}{y_n - y_0} d_y.$$

On peut considérer que $\alpha = \frac{x_n - x_0}{y_n - y_0}$ représente l'angle de la route par rapport à l'horizontale (comme le montre la figure 7.a. Dans le cas d'une route rectiligne, l'angle est bien évidemment une constante. Supposons maintenant que l'on modifie la valeur de cet angle. Par exemple si, sur le premier segment, on applique un angle α_1, puis

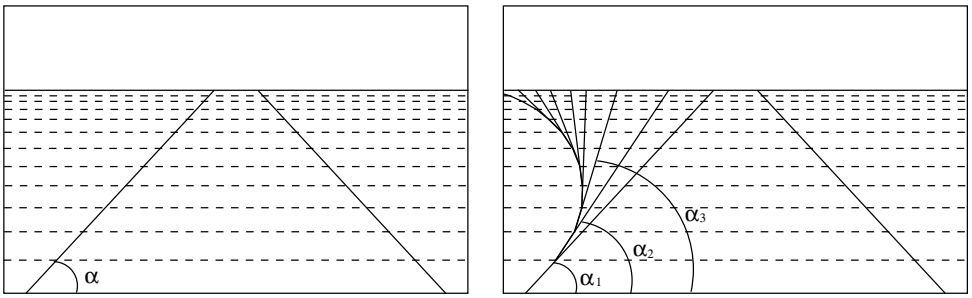


FIG. 7 – Les angles qui déterminent les bords gauches.

sur le deuxième un angle α_2, et ainsi de suite, la route prend une courbure vers la gauche, comme le montre la figure 7.b. Si l'on fait correspondre à chaque segment d'écran un segment de route et si, à ce segment on fait correspondre la différence d'angle par rapport au segment précédent[1] $\beta_i = \alpha_i - \alpha_{i-1}$, eh bien il est très simple de programmer l'affichage ci-dessus. En effet, il suffit d'appliquer l'algorithme suivant :

Au début de la procédure d'affichage, on détermine la position horizontale x_0 du bord gauche de la ligne la plus basse de l'écran ainsi que de celle (x_n) de la ligne la plus haute de l'écran. Ces positions permettent de calculer l'angle par défaut α. De plus, l'écran est découpé en un certain nombre de segments, qui correspondent à de petits segments de route. À chacun des segments de route, segment$_i$, est associé un incrément d'angle β_i. Pour calculer les positions des bords gauches, on utilise l'algorithme suivant :

```
angle = α, x = x0
pour tous les segments i (du bas vers le haut) faire
    pour toutes les ordonnées de pixels du segment du bas vers le haut faire
        abscisse du bord gauche du pixel = x
        x = x + angle
        angle = angle + βi
    fait
fait
```

Si β_i est négatif, on aura un tournant vers la gauche, si β_i est positif, on aura un tournant vers la droite et, bien évidemment, si β_i est nul, la route sera droite.

3 La mise en œuvre et le travail à rendre

Vous connaissez maintenant l'algorithme d'affichage de la route, la partie névralgique du programme à réaliser. Cela dit, rassurez-vous, je vais vous guider un peu dans la construction de votre logiciel. Pour cela, j'ai découpé le projet en un certain nombres d'étapes, correspondant grosso modo à celles que j'ai utilisées pour réaliser moi-même le projet. **À l'issue de chaque étape, vous me rendrez un listing de ce que vous aurez programmé**.

[1] On pourrait bien sûr associer à chaque segment de route l'angle α_i, mais le problème est que cet angle dépend manifestement de l'angle du segment précédent. Pour éviter cette dépendance, il suffit d'associer la différence entre l'angle courant et l'angle précédent, différence qui est constante.

3.1 Première étape : une ébauche d'affichage de la route (à rendre le 14/2/2002)

Votre logiciel 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 votre 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 intéresseront dans ce projet sont les suivants :

événement	cause
`WM_CREATE`	quand on crée la fenêtre
`WM_COMMAND`	quand on sélectionne un item dans un menu
`WM_PAINT`	repeint 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
`WM_KEYUP`	quand l'utilisateur relâche une touche

Le but de la première étape est de commencer à écrire la fonction d'affichage, le cœur de votre programme. Vous appelerez cette fonction dans le `case WM_PAINT`. Ainsi, elle sera appelée lorsque vous exécuterez votre programme.

Pour l'instant, il ne faut pas trop compliquer l'affichage, vous allez commencer par une route rectiligne et centrée. Le but de cette étape est de pouvoir afficher cette route. Dans un premier temps, vous n'affichez que la route en noir. Bien entendu, cela va de soi, votre affichage doit se faire par lignes horizontales comme décrit précédemment. Vous paramétrerez (`#define` au début du programme) les éléments définissant votre route (largeur de la plus haute ligne, de la plus basse, hauteur de la plus haute ligne, etc). Lorsque ceci est au point (vous devez obtenir un affichage similaire à la figure 8.a), sauvegardez votre projet et modifiez-le de manière à ne plus être centré sur la route. Vous devez alors obtenir un paysage similaire à la figure 8.b.

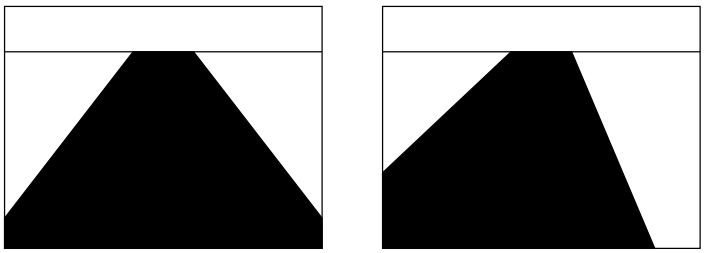


FIG. 8 – Une ébauche d'affichage.

Pour réaliser ces routes, vous aurez besoin des commandes `LineTo()` et `MoveToEx()` (dont la documentation est fournie en annexe).

3.2 Deuxième étape : affinage de l'affichage (à rendre le 14/3/2002)

On doit maintenant affiner un peu l'affichage, à savoir il faut dessiner les bandes latérales rouges et blanches ainsi que la ligne jaune au milieu de la route. Pour cela, il faut déjà pouvoir changer de couleur de trait. Les traits sont réalisés grâce à des stylos (des Pens). La première étape pour changer de couleur consiste à avoir à sa disposition des Pens. La fonction qui permet d'en créer est `CreatePen()` Vous allez donc utiliser cette fonction pour créer un stylo rouge, un jaune, un noir et un blanc. Une fois les stylos créés, on peut s'en servir en utilisant la fonction `SelectObject(hdc, pen)`. Bien entendu, vous ne créez pas les pens dans la fonction d'affichage, vous faites en sorte qu'ils le soient lors de l'initialisation de votre programme et, dans la procédure d'affichage, vous n'utilisez que le `SelectObject`. Pour pouvoir afficher les différentes bandes, il va falloir que vous découpiez l'écran en segments. À la fin de cette étape, vous devez obtenir des paysages similaires à celui de la figure 9.

3.3 Troisième étape : fin de l'affichage (à rendre le 11/4/2002)

Rapidement, vous allez incorporer le ciel et l'herbe. Pour cela, il suffit de dessiner deux grands rectangles pleins, l'un vert, l'autre bleu, avant de dessiner la route. Pour dessiner ces rectangles, vous utiliserez la fonction

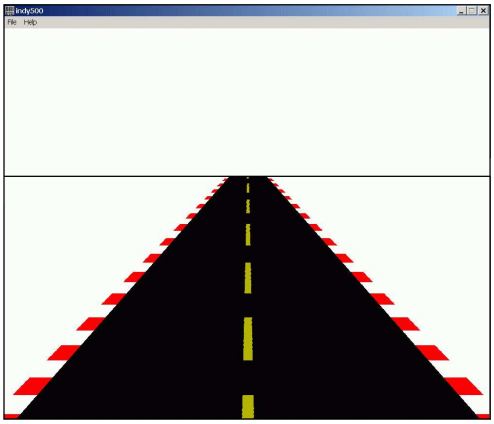


FIG. 9 – Affinage de la procédure d'affichage.

`FillRect()`. Pour déterminer la couleur de remplissage, on n'utilise pas un stylo, mais une brosse (Brush). La création d'une brosse se fait grâce à la fonction `CreateSolidBrush()`. La sélection d'une brosse se fait encore avec le SelectObject() : `SelectObject(hdc, brush)`.

Maintenant, la partie intéressante de cette étape : il faut inclure la notion de segment de route. Considérez que votre route est un ensemble de segments (100 segments par exemple), à chaque segment de route est associé un angle β_i. Modifiez votre procédure d'affichage de manière à tenir compte de ces β_i. Il faut évidemment tenir compte de la position du joueur sur la route pour savoir quels β_i prendre en compte. Vous considérerez que chaque segment représente une unité de longueur. Dans un premier temps, vous supposerez que la position du joueur est en multiples de cette unité puis, lorsque votre affichage sera au point, vous modifierez votre programme de manière à ce que la position du joueur puisse être n'importe quel réel (je suis aux 3/4 du premier segment par exemple). À l'issue de cette étape, vous devez obtenir un affichage similaire à celui de la figure 10

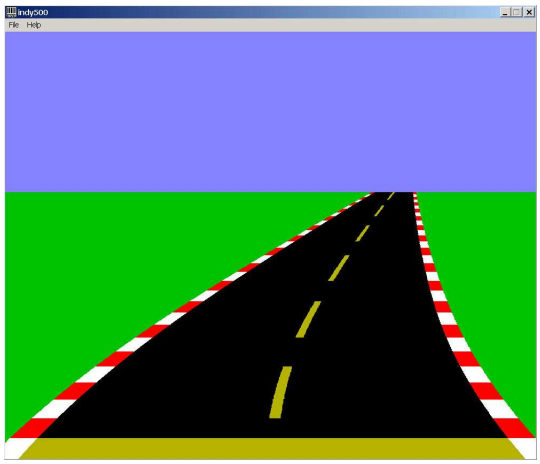


FIG. 10 – Fin de la procédure d'affichage.

3.4 Quatrième étape : gestion du joueur (à rendre le 2/5/2002)

L'introduction de la gestion de l'utilisateur va entraîner un certain nombre de transformations dans ce que vous avez déjà écrit. La première va être nécessaire si vous ne voulez pas que votre écran scintille. Jusqu'à maintenant, l'affichage était fixe, on pouvait donc dessiner dessus sans obtenir de problème de scintillement. Dans la version finale du jeu, vous allez mettre à jour le dessin un certain nombre de fois par secondes (probablement une vingtaine de fois). Si vous appelez la procédure d'affichage actuelle 20 fois par secondes, vous verrez apparaître ce scintillement. Pourquoi ? Simplement parce que, lors du réaffichage, vous commencez par imprimer les rectangles représentant le ciel et l'herbe, puis vous redessinez par dessus la route. Cette alternance herbe/route provoque le scintillement. Pour le supprimer, rien de plus simple en théorie : il suffit de tout redessiner sur un écran virtuel (non visible sur l'écran de votre ordinateur) puis, une fois le dessin achevé, de recopier cet écran virtuel sur le véritable écran. L'écran virtuel est un HDC. Voici la manière de le créer :

```
HDC display_hdc ; /* le device contexte de l'écran */
HDC internal_hdc ; /* le device contexte de l'écran virtuel */
HBITMAP internal_bitmap ; /* un bitmap stockant les pixels de l'écran virtuel */

/* on commence par récupérer le device contexte de la fenêtre actuelle */
display_hdc = GetDC(hWnd) ;
/* on crée l'écran virtuel */
internal_hdc = CreateCompatibleDC(display_hdc) ;
/* on crée un bitmap pour remplir l'écran virtuel */
internal_bitmap = CreateCompatibleBitmap(display_hdc, width, height) ;
/* enfin on associe le bitmap à l'écran virtuel */
SelectObject(internal_hdc, internal_bitmap) ;
```

Une fois que tout ceci est réalisé, `internal_hdc` est le device contexte de votre écran virtuel. Vous pouvez donc utiliser toutes vos primitives d'affichage (LineTo, etc) sur ce HDC, cela dessinera dans votre écran virtuel. Il vous reste alors à recopier l'écran virtuel sur votre véritable écran. C'est la fonction `BitBlt()` qui va vous le permettre :

```
BitBlt(display_hdc, 0, 0, window_width, window_height, internal_hdc, 0, 0, SRCCOPY);
```

Cf. l'annexe pour une description détaillée de la fonction `BitBlt()`.

Une fois que votre affichage est au point avec l'écran virtuel, vous pouvez passer à un réaffichage tous les $20^{\text{ème}}$ de seconde. Pour cela, il vous faut un timer. La création d'un timer se fait grâce à la fonction `SetTimer()` (cf. l'annexe). Une fois le timer créé, un événement `WM_TIMER` sera envoyé tous les $20^{\text{ème}}$ de seconde. Dans la fonction `WndProc()`, il ne vous reste plus qu'à rajouter un `case WM_TIMER`, et à lancer la fonction d'affichage dedans. Attention : pour que le timer se mette correctement en place, il semble qu'il faille laisser dans le `case WM_PAINT` les instructions `BeginPaint` et `EndPaint`. À titre indicatif, mon `case WM_PAINT` est le suivant :

```
case WM_PAINT:
      display_hdc = BeginPaint(hWnd, &ps);
      EndPaint(hWnd, &ps);
      break;
```

Testez si votre timer marche bien en faisant varier la position du joueur.

Lorsque votre timer et votre affichage fonctionnent, il convient de gérer effectivement les touches tapées par l'utilisateur. Lorsque l'utilisateur appuie sur une touche, un message `WM_KEYDOWN` est envoyé, qu'il convient de récupérer dans la fonction `WndProc()`. Dans le message envoyé, `wParam` contient la touche en question. Ce paramètre est égal à `VK_RIGHT` pour une flèche vers la droite, `VK_LEFT` pour une flèche vers la gauche, `VK_DOWN` pour une flèche vers le bas et `VK_UP` pour une flèche vers le haut. Lorsque l'utilisateur relâche une touche, c'est un message `WM_KEYUP` qui est envoyé. Bien sûr, vous pouvez modifier la position, la direction et la vitesse du joueur dans les `case` correspondant aux touches, mais je ne vous le conseille pas : en effet, une seule touche est prise en compte à la fois par ces événements. Par conséquent, le jeu ne peut pas prendre en compte le fait que, simultanément, vous accélérez et que vous tournez à gauche. Pour prendre en compte ce genre de configuration, j'ai utilisé des booléens qui indiquent si une touche est appuyée ou non. Lors d'un `WM_KEYDOWN`, le booléen passe à TRUE et lors d'un `WM_KEYUP`, il passe à FALSE. La mise à jour de la position, direction, vitesse, du joueur est alors réalisée chaque fois que le timer est réveillé.

3.5 Cinquième étape : quelques fioritures (à rendre le 23/5/2002)

Dans cette dernière étape, vous allez incorporer les derniers détails du jeu : l'affichage du chrono et de la voiture. Commençons par le temps et la vitesse, c'est plus simple : l'arc de cercle symbolisant le compteur de vitesse est réalisé grâce à la fonction `AngleArc()`. L'affichage du chrono est, quant à lui, réalisé avec `DrawText()`. Pour placer le texte là où vous voulez, vous pouvez utiliser la fonction `GetTextExtentPoint32()` qui vous calcule la taille en pixels d'une chaîne de caractères. Vous pouvez aussi changer la couleur de fond du texte avec la fonction `SetBkColor()`. Une manière assez précise de récupérer le temps est d'utiliser la fonction standard `clock()`.

L'affichage des voitures est un peu plus compliqué. Il faut d'abord charger en mémoire les cinq bitmaps (voiture1.bmp,...,voiture5.bmp). Une fois ces bitmaps en mémoire, on va les placer dans des HDC. Pour calculer la taille des bitmaps (ils sont tous de la même taille), on peut utiliser la fonction `GetObject()`. Voici ce que donne le chargement de la voiture 1 :

```
HBITMAP car_bitmap;
BITMAP  bitmap_info;
HDC     car_hdc;

/* chargement d'un bitmap contenant l'image de la voiture */
car_bitmap = (HBITMAP) LoadImage(hInst, "voiture1.bmp", IMAGE_BITMAP, 0, 0,
                                 LR_DEFAULTSIZE | LR_LOADFROMFILE);
/* création d'un HDC contenant cette image */
car_hdc = CreateCompatibleDC(display_hdc);
SelectObject(car_hdc, car_bitmap);
/* récupération de la taille de l'image */
if (!GetObject(car_bitmap, sizeof(BITMAP), &bitmap_info)) exit(1);
car_width  = bitmap_info.bmWidth;
car_height = bitmap_info.bmHeight;
```

Bien entendu, ce chargement n'est à faire qu'au lancement du programme. Ensuite, l'affichage de la voiture se fait grâce à la fonction `TransparentBlt()` :

```
TransparentBlt(internal_hdc, /* l'écran dans lequel on affiche la voiture */
      x_car, y_car, /* les coordonnées de la voiture dans l'écran */
      car_width, car_height, /* la taille de la voiture */
      car_hdc, /* le HDC contenant la voiture */
      0, 0, car_width, car_height, /* les coordonnées du rectangle contenant la voiture */
      0x4080ff); /* la couleur transparente (c'est une sorte d'orangé) */
```

3.6 Rapport final (à rendre le 20/6/2002)

Dans cette ultime étape, vous devez soigner votre listing, bien le présenter, vérifier l'indentation avec un notepad ou un wordpad (l'indentation de Visual C++ laisse parfois à désirer quand on sort le listing à l'imprimante). Je rappelle qu'il faut respecter les règles énoncées dans le memento. Sont pris en considération dans la note de projet les aspects suivants :

- la lisibilité du code (indentation, aération, commentaires) ;
- la beauté du code (j'aime les codes simples à lire) ;
- les bugs éventuels ;
- dans une insignifiante mesure, la beauté du jeu lui-même : ce qui m'intéresse dans un projet, ce n'est pas l'interface, mais ce qu'il y a derrière.

Vous me rendrez une version du programme sur disquette ainsi qu'une version papier du code. 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).

Enfin, vous devez me rendre un rapport papier m'indiquant ce que vous avez fait. Le rapport est destiné à permettre i) à un éventuel programmeur de compléter/mettre à jour votre logiciel ; ii) à un utilisateur de savoir comment utiliser votre logiciel. Il y a donc deux parties distinctes. En ce qui concerne la première, vous devez donner toutes les informations utiles pour comprendre rapidement vos algorithmes, le découpage du programme, les variables importantes/globales, bref tout ce qui permettrait à un autre programmeur de ne pas se prendre la tête pendant des jours à essayer de comprendre ce que vous avez fait.

4 Annexe : quelques indications et références

Afin de vous familiariser avec les fonctions, vous pourrez étudier avec profit 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()`.

Dans les sous-sections suivantes, vous trouverez les explications données à propos des fonctions graphiques suivantes par la MSDN library :

- AngleArc()
- BitBlt()
- CreateCompatibleDC()
- CreatePen()
- CreateSolidBrush()
- DeleteDC()
- DrawText()
- FillRect()
- GetDC()
- GetObject()
- GetTextExtentPoint32()
- LineTo()
- LoadImage()
- MoveToEx()
- ReleaseDC()
- SelectObject()
- SetBkColor()
- SetTimer()
- TransparentBlt()

4.1 AngleArc

The AngleArc function draws a line segment and an arc. The line segment is drawn from the current position to the beginning of the arc. The arc is drawn along the perimeter of a circle with the given radius and center. The length of the arc is defined by the given start and sweep angles.

```
BOOL AngleArc(
  HDC hdc,            // handle to device context
  int X,              // x-coordinate of circle's center
  int Y,              // y-coordinate of circle's center
  DWORD dwRadius,     // circle's radius
  FLOAT eStartAngle,  // arc's start angle
  FLOAT eSweepAngle   // arc's sweep angle
);
```

Parameters :

hdc	Handle to a device context.
X	Specifies the logical x-coordinate of the center of the circle.
Y	Specifies the logical y-coordinate of the center of the circle.
dwRadius	Specifies the radius, in logical units, of the circle. This value must be positive.
eStartAngle	Specifies the start angle, in degrees, relative to the x-axis.
eSweepAngle	Specifies the sweep angle, in degrees, relative to the starting angle.

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 :

The AngleArc function moves the current position to the ending point of the arc.
The arc drawn by this function may appear to be elliptical, depending on the current transformation and mapping mode. Before drawing the arc, AngleArc draws the line segment from the current position to the beginning of the arc.
The arc is drawn by constructing an imaginary circle around the specified center point with the specified radius. The starting point of the arc is determined by measuring counterclockwise from the x-axis of the circle by the number of degrees in the start angle. The ending point is similarly located by measuring counterclockwise from the starting point by the number of degrees in the sweep angle.
If the sweep angle is greater than 360 degrees, the arc is swept multiple times.
This function draws lines by using the current pen. The figure is not filled.

4.2 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.

4.3 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.

4.4 CreatePen

The CreatePen function creates a logical pen that has the specified style, width, and color. The pen can subsequently be selected into a device context and used to draw lines and curves.

```
HPEN CreatePen(
  int fnPenStyle,    // pen style
  int nWidth,        // pen width
  COLORREF crColor   // pen color
);
```

Parameters :

<table>
<tr><td>fnPenStyle</td><td>Specifies the pen style. It can be any one of the following values.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>PS_SOLID</td><td>The pen is solid.</td></tr>
<tr><td>PS_DASH</td><td>The pen is dashed. This style is valid only when the pen width is one or less in device units.</td></tr>
<tr><td>PS_DOT</td><td>The pen is dotted. This style is valid only when the pen width is one or less in device units.</td></tr>
<tr><td>PS_DASHDOT</td><td>The pen has alternating dashes and dots. This style is valid only when the pen width is one or less in device units.</td></tr>
<tr><td>PS_DASHDOTDOT</td><td>The pen has alternating dashes and double dots. This style is valid only when the pen width is one or less in device units.</td></tr>
<tr><td>PS_NULL</td><td>The pen is invisible.</td></tr>
<tr><td>PS_INSIDEFRAME</td><td>The pen is solid. When this pen is used in any GDI drawing function that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.</td></tr>
</table></td></tr>
<tr><td>nWidth</td><td>Specifies the width of the pen, in logical units. If nWidth is zero, the pen is a single pixel wide, regardless of the current transformation.
CreatePen returns a pen with the specified width bit with the PS_SOLID style if you specify a width greater than one for the following styles : PS_DASH, PS_DOT, PS_DASHDOT, PS_DASHDOTDOT.</td></tr>
<tr><td>crColor</td><td>Specifies a color reference for the pen color.</td></tr>
</table>

Return Values :

If the function succeeds, the return value is a handle that identifies a logical pen.
If the function fails, the return value is NULL.
Windows NT : To get extended error information, call GetLastError.

Remarks :

After an application creates a logical pen, it can select that pen into a device context by calling the SelectObject function. After a pen is selected into a device context, it can be used to draw lines and curves.
If the value specified by the nWidth parameter is zero, a line drawn with the created pen always is a single pixel wide regardless of the current transformation.
If the value specified by nWidth is greater than 1, the fnPenStyle parameter must be PS_NULL, PS_SOLID, or PS_INSIDEFRAME.
If the value specified by nWidth is greater than 1 and fnPenStyle is PS_INSIDEFRAME, the line associated with the pen is drawn inside the frame of all primitives except polygons and polylines.
If the value specified by nWidth is greater than 1, fnPenStyle is PS_INSIDEFRAME, and the color specified by the crColor parameter does not match one of the entries in the logical palette, the system draws lines by using a dithered color. Dithered colors are not available with solid pens.
When you no longer need the pen, call the DeleteObject function to delete it.
ICM : No color management is done at creation. However, color management is performed when the pen is selected into an ICM-enabled device context.

4.5 CreateSolidBrush

The CreateSolidBrush function creates a logical brush that has the specified solid color.

```
HBRUSH CreateSolidBrush(
  COLORREF crColor   // brush color value
);
```

Parameters :

crColor	Specifies the color of the brush.

Return Values :

If the function succeeds, the return value identifies a logical brush.
If the function fails, the return value is NULL.
Windows NT : To get extended error information, call GetLastError.

Remarks :

A solid brush is a bitmap that the system uses to paint the interiors of filled shapes.
After an application creates a brush by calling CreateSolidBrush, it can select that brush into any device context by calling the SelectObject function.
ICM : No color is done at brush creation. However, color management is performed when the brush is selected into an ICM-enabled device context.

4.6 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.

4.7 DrawText

The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).

```
int DrawText(
  HDC hDC,          // handle to device context
  LPCTSTR lpString, // pointer to string to draw
  int nCount,       // string length, in characters
  LPRECT lpRect,    // pointer to struct with formatting dimensions
  UINT uFormat      // text-drawing flags
);
```

Parameters :

hDC	Handle to the device context.
lpString	Pointer to the string to be drawn. If the nCount parameter is −1, the string must be null-terminated. If uFormat includes DT_MODIFYSTRING, the function could add up to four additional characters to this string. The buffer containing the string should be large enough to accommodate these extra characters.
nCount	Specifies the number of characters in the string. If nCount is -1, then the lpString parameter is assumed to be a pointer to a null-terminated string and DrawText computes the character count automatically. Windows 95/98 : The number of characters to be drawn many not exceed 8192.
lpRect	Pointer to a RECT structure that contains the rectangle (in logical coordinates) in which the text is to be formatted.
uFormat	Specifies the method of formatting the text. It can be any combination of the following values.

Value	Description
DT_BOTTOM	Justifies the text to the bottom of the rectangle. This value must be combined with DT_SINGLELINE.
DT_CALCRECT	Determines the width and height of the rectangle. If there are multiple lines of text, DrawText uses the width of the rectangle pointed to by the lpRect parameter and extends the base of the rectangle to bound the last line of text. If there is only one line of text, DrawText modifies the right side of the rectangle so that it bounds the last character in the line. In either case, DrawText returns the height of the formatted text but does not draw the text.
DT_CENTER	Centers text horizontally in the rectangle.
DT_EDITCONTROL	Duplicates the text-displaying characteristics of a multiline edit control. Specifically, the average character width is calculated in the same manner as for an edit control, and the function does not display a partially visible last line.
DT_END_ELLIPSIS	Replaces part of the given string with ellipses, if necessary, so that the result fits in the specified rectangle. The given string is not modified unless the DT_MODIFYSTRING flag is specified.
DT_EXPANDTABS	Expands tab characters. The default number of characters per tab is eight. The DT_WORD_ELLIPSIS and DT_END_ELLIPSIS values cannot be used with the DT_EXPANDTABS value.
DT_LEFT	Aligns text to the left.
DT_MODIFYSTRING	Modifies the given string to match the displayed text. This flag has no effect unless the DT_END_ELLIPSIS flag is specified.
DT_NOCLIP	Draws without clipping. DrawText is somewhat faster when DT_NOCLIP is used.
DT_NOFULLWIDTHCHARBREAK	Windows NT 5.0 and later, Windows 98 : Prevents a line break at a DBCS (double-wide character string), so that the line breaking rule is equivalent to SBCS strings. For example, this can be used in Korean windows, for more readability of icon labels. This is only effective if DT_WORDBREAK is specified.
DT_RIGHT	Aligns text to the right.
DT_SINGLELINE	Displays text on a single line only. Carriage returns and linefeeds do not break the line.
DT_TABSTOP	Sets tab stops. Bits 15–8 (high-order byte of the low-order word) of the uFormat parameter specify the number of characters for each tab. The default number of characters per tab is eight.The DT_CALCRECT and DT_NOCLIP values cannot be used with the DT_TABSTOP value.
DT_TOP	Top-justifies text (single line only).
DT_VCENTER	Centers text vertically (single line only).
DT_WORDBREAK	Breaks words. Lines are automatically broken between words if a word would extend past the edge of the rectangle specified by the lpRect parameter. A carriage return-linefeed sequence also breaks the line.
DT_WORD_ELLIPSIS	Truncates text that does not fit in the rectangle and adds ellipses.

Return Values :

If the function succeeds, the return value is the height of the text. If `DT_VCENTER` or `DT_BOTTOM` is specified, the return value is the offset from `lpRect->top` to the bottom of the drawn text.
If the function fails, the return value is zero.
Windows NT : To get extended error information, call GetLastError.

Remarks :

The DrawText function uses the device context's selected font, text color, and background color to draw the text. Unless the `DT_NOCLIP` format is used, DrawText clips the text so that it does not appear outside the specified rectangle. All formatting is assumed to have multiple lines unless the `DT_SINGLELINE` format is specified.
If the selected font is too large for the specified rectangle, the DrawText function does not attempt to substitute a smaller font.

4.8 FillRect

The FillRect function fills a rectangle by using the specified brush. This function includes the left and top borders, but excludes the right and bottom borders of the rectangle.

```
int FillRect(
  HDC hDC,           // handle to device context
  CONST RECT *lprc,  // pointer to structure with rectangle
  HBRUSH hbr         // handle to brush
);
```

Parameters :

hDC	Handle to the device context.
lprc	Pointer to a RECT structure that contains the logical coordinates of the rectangle to be filled.
hbr	Handle to the brush used to fill the rectangle.

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 :

The brush identified by the hbr parameter may be either a handle to a logical brush or a color value. If specifying a handle to a logical brush, call one of the following functions to obtain the handle : CreateHatchBrush, CreatePatternBrush, or CreateSolidBrush. Additionally, you may retrieve a handle to one of the stock brushes by using the GetStockObject function. If specifying a color value for the hbr parameter, it must be one of the standard system colors (the value 1 must be added to the chosen color). For example :

```
FillRect(hdc, &rect, (HBRUSH) (COLOR_WINDOW+1));
```

For a list of all the standard system colors, see GetSysColor.

When filling the specified rectangle, FillRect does not include the rectangle's right and bottom sides. GDI fills a rectangle up to, but not including, the right column and bottom row, regardless of the current mapping mode.

4.9 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.

4.10 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 colspan="3">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 colspan="3">Specifies the number of bytes of information to be written to the buffer.</td></tr>
<tr><td>lpvObject</td><td colspan="3">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>
</table>
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>

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.

4.11 GetTextExtentPoint32

The GetTextExtentPoint32 function computes the width and height of the specified string of text.

```
BOOL GetTextExtentPoint32(
  HDC hdc,           // handle to device context
  LPCTSTR lpString,  // pointer to text string
  int cbString,      // number of characters in string
  LPSIZE lpSize      // pointer to structure for string size
);
```

Parameters :

hdc	Handle to the device context.
lpString	Pointer to the string of text. The string does not need to be zero-terminated, since cbString specifies the length of the string.
cbString	Specifies the number of characters in the string. Windows 95/98 : This value may not exceed 8192.
lpSize	Pointer to a SIZE structure in which the dimensions of the string are to be returned.

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 :

The GetTextExtentPoint32 function uses the currently selected font to compute the dimensions of the string. The width and height, in logical units, are computed without considering any clipping.
Because some devices kern characters, the sum of the extents of the characters in a string may not be equal to the extent of the string.
The calculated string width takes into account the intercharacter spacing set by the SetTextCharacterExtra function.

4.12 LineTo

The LineTo function draws a line from the current position up to, but not including, the specified point.

```
BOOL LineTo(
  HDC hdc,    // device context handle
  int nXEnd,  // x-coordinate of line's ending point
  int nYEnd   // y-coordinate of line's ending point
);
```

Parameters :

hdc	Handle to a device context.
nXEnd	Specifies the x-coordinate of the line's ending point.
nYEnd	Specifies the y-coordinate of the line's ending point.

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 :

The coordinates of the line's ending point are specified in logical units.
The line is drawn by using the current pen and, if the pen is a geometric pen, the current brush.
If LineTo succeeds, the current position is set to the specified ending point.

4.13 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>

	Value	Meaning
fuLoad	LR_SHARED	Shares the image handle if the image is loaded multiple times. If LR_SHARED is not set, a second call to LoadImage for the same resource will load the image again and return a different handle. When you use this flag, the system will destroy the resource when it is no longer needed. Do not use LR_SHARED for images that have non-standard sizes, that may change after loading, or that are loaded from a file. Windows 95/98 : The function finds the first image with the requested resource name in the cache, regardless of the size requested.
	LR_VGACOLOR	Uses true VGA colors.

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.

4.14 MoveToEx

The MoveToEx function updates the current position to the specified point and optionally returns the previous position.

```
BOOL MoveToEx(
  HDC hdc,          // handle to device context
  int X,            // x-coordinate of new current position
  int Y,            // y-coordinate of new current position
  LPPOINT lpPoint   // pointer to old current position
);
```

Parameters :

hdc	Handle to a device context.
X	Specifies the x-coordinate of the new position, in logical units.
Y	Specifies the y-coordinate of the new position, in logical units.
lpPoint	Pointer to a POINT structure in which the previous current position is stored. If this parameter is a NULL pointer, the previous position is not returned.

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 :

The MoveToEx function affects all drawing functions.

4.15 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.

4.16 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 :

<table>
<tr><td>hdc</td><td colspan="2">Handle to the device context.</td></tr>
<tr><td>hgdiobj</td><td colspan="2">Handle to the object to be selected. The specified object must have been created by using one of the following functions.</td></tr>
<tr><td></td><td>Object</td><td>Functions</td></tr>
<tr><td></td><td>Bitmap</td><td>CreateBitmap, CreateBitmapIndirect, CreateCompatibleBitmap, CreateDIBitmap, CreateDIBSection (Bitmaps can be selected for memory device contexts only, and for only one device context at a time.)</td></tr>
<tr><td></td><td>Brush</td><td>CreateBrushIndirect, CreateDIBPatternBrush, CreateDIBPatternBrushPt, CreateHatchBrush, CreatePatternBrush, CreateSolidBrush Font CreateFont, CreateFontIndirect</td></tr>
<tr><td></td><td>Pen</td><td>CreatePen, CreatePenIndirect</td></tr>
<tr><td></td><td>Region</td><td>CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreateRectRgn, CreateRectRgnIndirect</td></tr>
</table>

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.

4.17 SetBkColor

The SetBkColor function sets the current background color to the specified color value, or to the nearest physical color if the device cannot represent the specified color value.

```
COLORREF SetBkColor(
  HDC hdc,           // handle of device context
  COLORREF crColor   // background color value
);
```

Parameters :

hdc	Handle to the device context.
crColor	Specifies the new background color.

Return Values :

If the function succeeds, the return value specifies the previous background color as a COLORREF value.
If the function fails, the return value is CLR_INVALID.
Windows NT : To get extended error information, call GetLastError.

Remarks :

This function fills the gaps between styled lines drawn using a pen created by the CreatePen function ; it does not fill the gaps between styled lines drawn using a pen created by the ExtCreatePen function. The SetBKColor function also sets the background colors for TextOut and ExtTextOut.
If the background mode is OPAQUE, the background color is used to fill gaps between styled lines, gaps between hatched lines in brushes, and character cells. The background color is also used when converting bitmaps from color to monochrome and vice versa.

4.18 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.

4.19 TransparentBlt

The TransparentBlt 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 TransparentBlt(
  HDC hdcDest,
  int nXOriginDest,
  int nYOriginDest,
  int nWidthDest,
  int hHeightDest,
  HDC hdcSrc,
  int nXOriginSrc,
  int nYOriginSrc,
  int nWidthSrc,
  int nHeightSrc,
  UINT crTransparent
);
```

Parameters :

hdcDest	Handle to the destination device context.
nXOriginDest	Specifies the x-coordinate, in logical units, of the upper-left corner of the destination rectangle.
nYOriginDest	Specifies the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.
nWidthDest	Specifies the width, in logical units, of the destination rectangle.
hHeightDest	Handle to the height, in logical units, of the destination rectangle.
hdcSrc	Handle to the source device context.
nXOriginSrc	Specifies the x-coordinate, in logical units, of the source rectangle.
nYOriginSrc	Specifies the y-coordinate, in logical units, of the source rectangle.
nWidthSrc	Specifies the width, in logical units, of the source rectangle.
nHeightSrc	Specifies the height, in logical units, of the source rectangle.
crTransparent	The RGB color in the source bitmap to treat as transparent.

Return Values :

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

Remarks :

The TransparentBlt function supports all formats of source bitmaps. However, for 32 bpp bitmaps, it just copies the alpha value over. Use AlphaBlend to specify 32 bits-per-pixel bitmaps with transparency.
If the source and destination rectangles are not the same size, the source bitmap is stretched to match the destination rectangle. When the SetStretchBltMode function is used, the iStretchMode modes of BLACKONWHITE and WHITEONBLACK are converted to COLORONCOLOR for the TransparentBlt function.
The destination device context specifies the transformation type for the destination coordinates. The source device context specifies the transformation type for the source coordinates.
TransparentBlt does not mirror a bitmap if either the width or height, of either the source or destination, is negative.