3D GameStudio

Les Ateliers de Animation 2D

pour A5 Engine 5.10
par Alain Brégeon September 2001

Les dernières nouvelles, les démonstrations, les mises à jour et les outils, aussi bien que le Magazine des Utilisateurs, le Forum des Utilisateurs et le concours annuel sont disponibles à la page principale GameStudio **http://www.3dgamestudio.com**.

Contente

Avant propos de la part de l'auteur

Cher Lecteur,

J'ai produit cet atelier pour vous aider à répondre à la question "Comment faire un jeu d'**animation 2D** avec 3DGameStudio ?". Cet atelier utilise des possibilités disponibles à partir de la version (**4.25**) ou supérieure.

Cet atelier, comme les autres ateliers avant cela, vise surtout les utilisateurs qui ont un peu d'expérience de 3DGameStudio. Je suppose que vous avez travaillé les différents tutoriaux et savez comment employer les outils (WED, MED et WDL).

Ce texte complète la documentation qui va avec 3DGameStudio, et ne la remplace pas. Si quelque chose dans cet atelier est peu clair lisez s'il vous plait les manuels qui sont fournis avec 3DGameStudio. Je fais par avance des excuses pour des formulations que vous trouveriez peu claires, un code défectueux, des erreurs ou des omissions.

J'espère que vous trouverez cet atelier informatif et agréable.

Alain Brégeon
mailto:alainbregeon@hotmail.com

Obtenez la dernière version

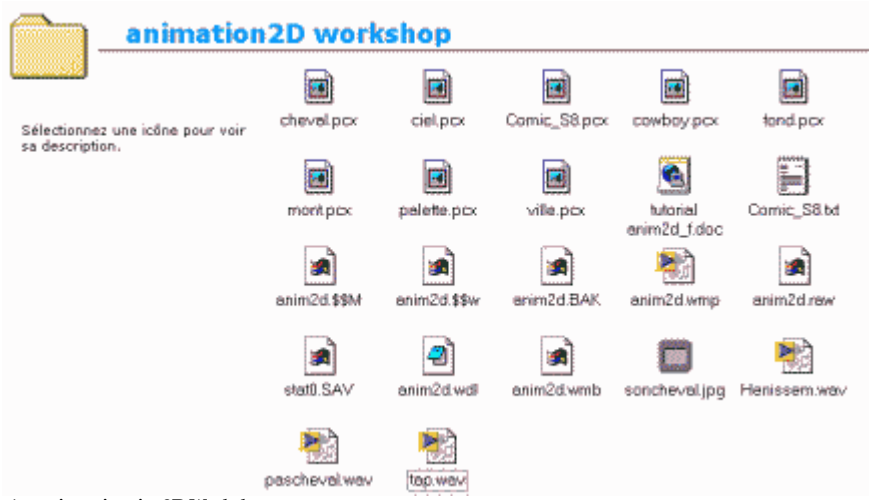
Avant de commencer, assurez-vous d'avoir la dernière version de 3DGameStudio (**4.25** ou au-dessus).

Préparez votre workspace

Créez un dossier appelé "animation2d Workshop" dans votre dossier GSTUDIO. C'est le répertoire ou vous stockerez tous les éléments du jeu.

La première chose que nous allons ajouter à notre dossier est le modèle de jeu. Si vous ne les avez pas déjà, allez à la page de téléchargement de Conitec (http: // www.conitec.net/a4update.htm) et récupérez le niveau de Fighting. Décompressez le contenu dans votre dossier. Votre dossier doit maintenant contenir au moins les fichiers:

```
cheval.pcx
ciel.pcx
comic_s8.pcx
fond.pcx
mont.pcx
palette.pcx
ville.pcx
tutorial.doc
comic_s8.txt
henissem.wav
pascheval.wav
tap.wav
```



répertoire animation2D Workshop

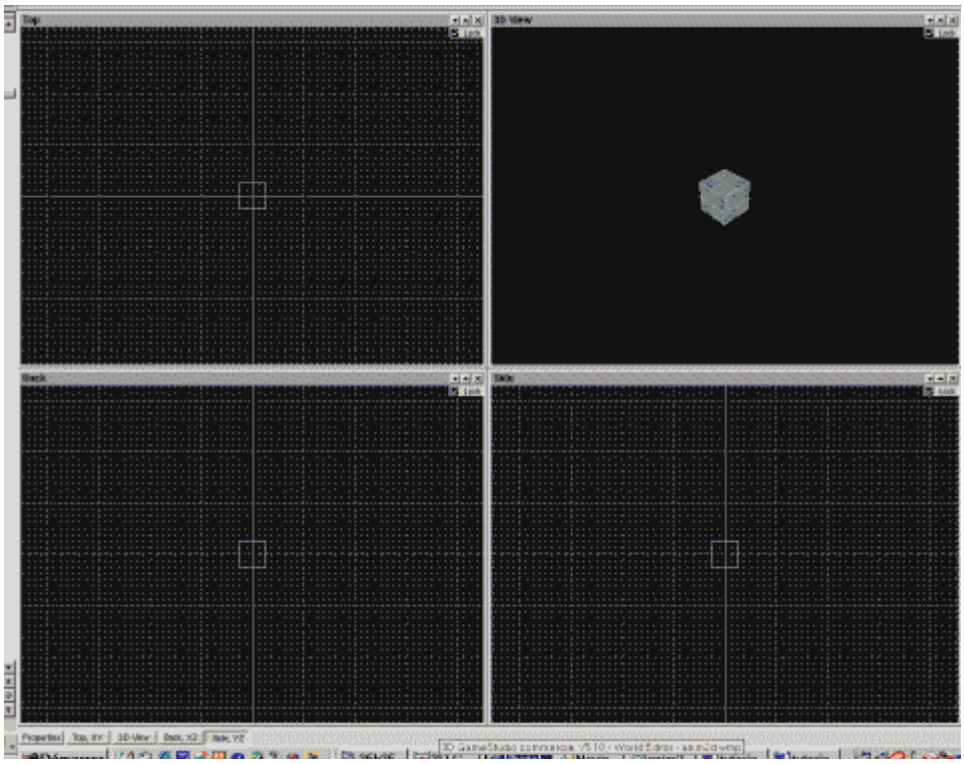
Création du niveau

Notre niveau sera très simple car inutilisé. Mais comme il faut au moins un objet de créé pour pouvoir exécuter le moteur 2 D nous créons la carte suivante :

Ouvrez WED et choisissez **File-> New**

Choisissez **Add Primitive - > Cube(small)**.

Nous appliquons la texture par défaut, nous sauvegardons le niveau, nous l'appelons **`anim2d`**, le compilons et l'exécutons. Il ne se passe rien c'est normal.



Création de votre script

Créez un scénario pour votre niveau. Ouvrez votre fenêtre de Propriétés de Carte (**File-> MapProperties**) et appuyez sur bouton **new**. Le bouton à côté du Scénario doit changer de **ndef** à **anim2d.wdl**.

Ouvrez votre dossier de niveau sélectionnez et ouvrez (double-clic) le fichier **anim2d.wdl**. Si Windows demande quelle application employer pour ouvrir le fichier, choisissez "bloc-notes" (ou n'importe quel éditeur de texte simple).

Vous devez remarquer que le jeu typique, "modèle" a été créé pour vous. C'est suffisant pour la plupart des projets mais nous voulons faire quelque chose de plus avancé cette fois. Allons-y et sélectionnez tout (dans le bloc-notes de Microsoft employez **Edit-> select All**) et cliquez sur la touche "suppr". Maintenant nous allons tout recommencer à zéro.

Ajout d'un chemin

Commençons en définissant les chemins que notre programme va employer. Les chemins sont employés pour dire au moteur où il peut placer les fichiers employés dans notre projet (des images, des sons, d'autres scénarios, etc.). Le dossier ou nous sommes ("anim2d Workshop") est déjà inclus, aussi, dans notre cas, nous devons seulement ajouter le répertoire **template**.

Tapez la ligne suivante :

```
path  "..\\template";   // chemin du sous répertoire templates de wdl
```

Notez que tous les chemins sont relatifs à notre dossier de niveau. La ligne ci-dessus dit ceci,

"remonte d'un niveau (" ..") et redescend dans le dossier template (**\\ template**)".

Ajoutez les "include"

Après que nous ayons créé nos chemins nous devons ajouter nos fichiers **include**. Tapez sous **path**:

```
include <movement.wdl>;  // libraries of WDL functions
include <messages.wdl>;
include <menu.wdl>;      // menu must be included BEFORE doors and weapons
include <particle.wdl>; // remove when you need no particles
```

La commande **include** dit au moteur de remplacer cette ligne avec le contenu du fichier entre (<... >). C'est comme si nous étions allés chercher le fichier en question, avions copié tout son code, et l'avions collé dans le scénario.

C'est un outil puissant parce qu'il nous permet de réutiliser le code d'autres projets et permet de faire des mises à jour dans les fichiers inclus sans devoir récrire votre code. Par exemple, la mise à jour 4.19 inclut du code pour permettre au joueur de nager dans l'eau. Ainsi n'importe quel projet qui inclut **movement.wdl** peut employer ce nouveau code pour la nage.

Comme pour les chemins, l'ordre des lignes **include** est important. Puisque quelques scénarios utilisent des valeurs qui sont déclarées dans d'autres scénarios. Par exemple : **actors.wdl** emploie la variable, **force** qui est déclarée dans **movement.wdl**. Si nous mettons **actors.wdl** avant **movement.wdl** nous obtiendrions des erreurs.

C'est bien d'INCLURE des scénarios même si vous n'employez aucune de leurs particularités dans votre code. La plupart des fichiers dans template sont interdépendants l'un sur l'autre aussi si vous faites des projets pour utiliser l'un d'entre eux, vous devez les INCLURE tous pour être sûrs. L'exception à cette règle est le scénario **venture.wdl**, qui n'est employé par aucun des autres scénarios, mais emploie chacun d'eux.

Les valeurs de départ du moteur

Maintenant nous allons mettre quelques valeurs importantes qui aideront à décider comment le simulateur sera montré. Ces valeurs déterminent la résolution, la profondeur des couleurs, le taux de rafraîchissement et l'éclairage. Ajoutez les lignes suivantes au-dessous des lignes **include**:

```
 // Valeurs de départ du moteur
ifdef lores;
var video_mode = 4; // 320x240
ifelse;
var video_mode = 6; // 640x480
endif;
var video_depth = 16; // D3D, 16 bit resolution
var fps_max = 50; // 50 fps max
```

Nos variables de jeux

C'est ici que nous entrerons nos variables de jeux au fur et à mesure de nos besoins
```
//our skills **************************************************
```

L'affichage du logo A4/A5

Nous préparons l'affichage du logo qui se trouve dans le répertoire template

```
////////////////////////////////////////////////////////////
// define a splash screen with the required A4/A5 logo
bmap splashmap = <logodark.bmp>; // the default A5 logo in templates
panel splashscreen { bmap = splashmap; flags = refresh,d3d; }
```

La Fonction Principale "Main"

Dans n'importe quel projet vous avez besoin de la fonction **`main`** (principale). C'est la première fonction à être appelée lorsque le programme démarre. Dans la plupart des cas la fonction principale est très simple, la notre ne fera pas exception. Entrez s'il vous plaît les lignes suivantes (sous votre dernière ligne) :

```
function main()
{
   fps_max = 50;
   warn_level = 2;    // announce bad texture sizes and bad wdl code

// center the splash screen for non-640x480 resolutions
   splashscreen.pos_x = (screen_size.x - bmap_width(splashmap))/2;
   splashscreen.pos_y = (screen_size.y - bmap_height(splashmap))/2;
// set it visible
   splashscreen.visible = on;
// wait 3 frames (for triple buffering) until it is renderight and flipped to the foreground
   wait(3);

// now load the level
   load_level (<anim2d.wmb>);
// wait the requiright second, then switch the splashscreen off.
   waitt(16);
   splashscreen.visible = off;
   bmap_purge(splashmap);   // remove logo bitmap from video memory

// load some global variables, like sound volume
   load_status();

   game();
}
function game()
{
}
```

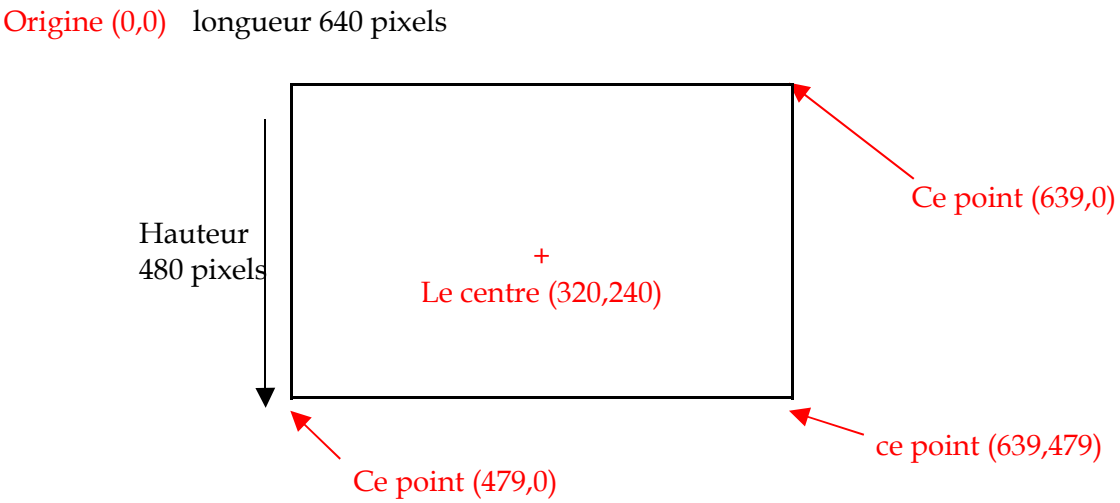
Chacune de ces lignes est commentée et n'appelle pas d'explications supplémentaires.

Le B.A-BA de la 2D

Pour avancer dans l'utilisation de la 2D il est important de connaître un certain nombre de choses que nous allons décrire dés à présent :

Les coordonnées

La fenêtre d'affichage est représentée par ses dimensions en pixels, longueur et hauteur, par exemple 640 x 480. La première particularité à connaître est la position de l'origine qui ne se trouve pas en bas à gauche comme on pourrait s'y attendre mais dans l'angle supérieur gauche. Chaque point pouvant être représenté sous la forme (x,y) :

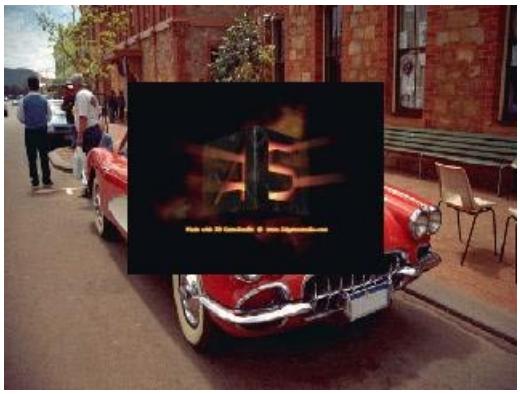


La palettisation:

Pour expliquer ce concept en apparence un peu barbare nous allons afficher une image que vous connaissez bien, le logo du moteur A5 :

Puis nous prenons 2 images splendides :

Et nous copions / collons notre logo dans chacune des images, nous obtenons ceci :



Le logo n'a plus aussi fière allure et si vous ne voyez pas la différence, mettons la voiture dans l'image du lac et vous comprendrez encore mieux :

Explications :
Tout d'abord il faut savoir que ces phénomènes n'arrivent qu'en mode 8 bits. Mais qu'est-ce que le mode 8 bits me direz-vous, c'est vrai qu'une fois que l'on a compris cela, tout devient plus clair.

Dans le mode 8 bits, chaque point de notre image (pixel) est représenté par un nombre qui peut prendre les valeurs de 0 à 255 (en mode binaire 00000000 à 11111111 donc 8 bits). Chacune de ces valeurs correspond à l'index d'une table qui contient 3 valeurs par index. Ces 3 valeurs correspondent aux 3 composantes de couleur (R, V, et B). Cette table s'appelle **palette**.

Prenons un petit exemple en utilisant le premier point de notre image (pour plus de clarté nous

utiliserons les 7 premières valeurs de notre palette) :

Position	0	1	2	3	4	5	6	…	255
Couleur									

Position	0	1	2	3	4	5	6	…	255
Couleur									

Mon premier point, celui qui se trouve en haut à gauche de mon image à la valeur 4, ce qui correspond à 'affiche la couleur que l'on trouve à la position 4 de la palette'.

Première palette :

Notre premier point qui a la couleur 4 sera donc :

Deuxième palette :

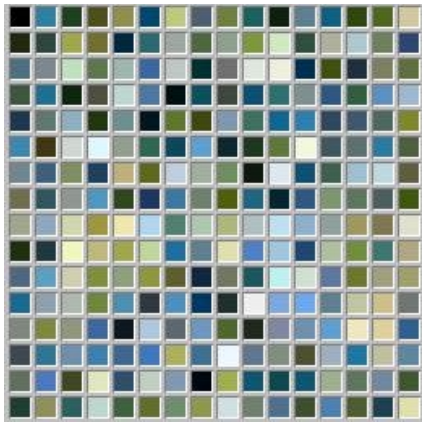
Notre premier point qui a la couleur 4 sera donc :

Mon image n'a pas changé, son premier point est toujours 4, mais sa représentation graphique a changé car dans le premier cas la palette dit à chaque point 4 je donne la couleur en correspondance alors que la deuxième palette dit à chaque point 4 je donne la couleur en correspondance.

Rappel :
Dans notre moteur la valeur 0 de notre palette doit toujours être le noir (R = 0, V = 0 et B = 0) et la valeur 255 de notre palette doit toujours être le blanc. (R = 255, V = 255 et B = 255)

Il est bien entendu que nous travaillerons le plus possible avec des images en 16 ou 32 bits mais comme notre jeu doit s'adapter à tous les environnements nous savons qu'il est susceptible de tourner en mode 8 bits sur certaines vieilles machines.

On comprend tout de suite qu'en collant la voiture sur l'image du lac, elle perd sa couleur car il n'y a aucune composante rouge dans notre palette lac.Voici en exemple la palette utilisée pour le lac et celle utilisée pour la voiture :


Palette du lac

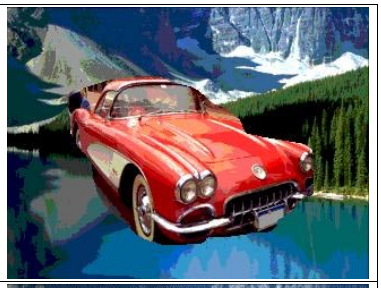
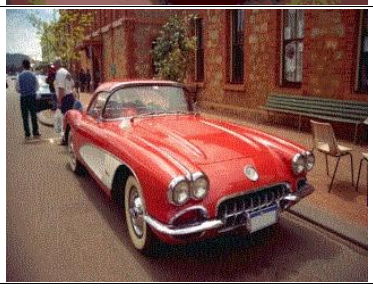
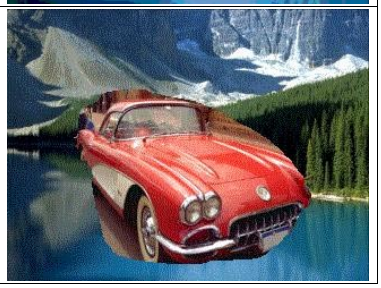

Palette de la voiture

Alors comment faire pour concilier l'inconciliable ?

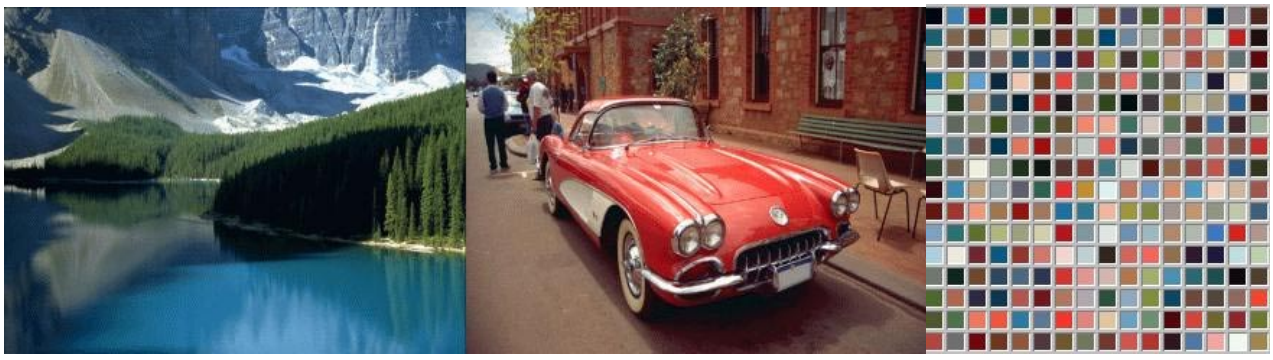
Première solution, travailler avec les palettes standard de Windows, mais c'est rarement génial,

voici le résultat avec la palette appelée 666 que voici :

Mon logiciel de dessin me permet de choisir mode de tramage ou pas, voici les 2 :

Sans tramage			
Avec tramage			

Une autre solution consiste à regrouper les 2 images dans une seule en mode 16 bits et de demander une conversion 8 bits optimisée. L'optimisation regarde toutes les couleurs utilisées et génère la meilleure palette possible. Voici le résultat :



L'overlay:

Autre notion très importante à connaître pour la réalisation de vos jeux 2 D. Le manuel WDL nous donne l'explication suivante : Si ce drapeau est mis, la couleur 0 (obligatoirement noir) de l'image ne sera pas dessinée…

En clair j'ai une image que je veux faire apparaître sur mon image de fond, imaginons une mongolfière survolant mon lac :

lorsque je colle cette montgolfière voici ce que j'obtiens:

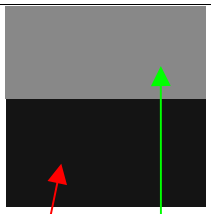
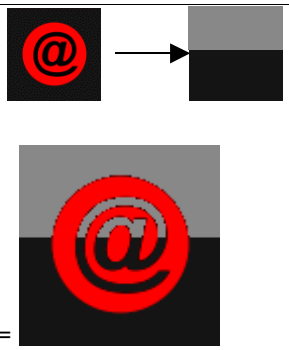
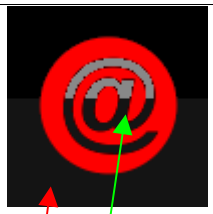
Pas très joli, nous allons donc utiliser la fonction overlay et pour cela nous devons remplacer les surfaces à cacher (en l'occurrence le blanc) par du noir, voici le résultat :

C'est beaucoup mieux.
Mais que ce serait-il passé s'il y avait du noir sur ma mongolfière, imaginons une publicité sur le ballon, vous devinez par avance le résultat

le @ qui était en noir est devenu transparent et ça c'est pas terrible.

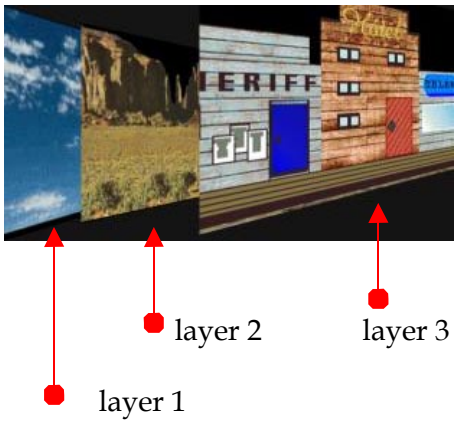
Petite astuce d'un vieux combattant de l'informatique :

Dans mon logiciel de dessin, je prends mon image montgolfière et je la superpose sur un fond gris foncé en demandant une transparence de mon noir, donc toutes les surfaces noires, utiles pour la transparence et les autres vont se trouver gris foncé. Ensuite je remets du noir sur mes autres qui doivent être transparente. Suivez en image. Pour plus de clarté le gris foncé est clair sur les dessins afin que vous voyiez bien l'explication :

			
Le noir est noir (r = 0, v = 0 et b = 0)	Tout le fond est normalement gris foncé C'est à dire que le noir que vous voyez n'est pas noir noir mais gris foncé (r = 20, v = 20 et b = 20)	Je demande la transparence pour mon noir, je vois donc mon fond gris (foncé) par transparence. Je fusionne les deux.	Nous remettons du noir (0,0,0) à l'extérieur, le noir intérieur étant gris foncé (20,20,20) mais qui voit la différence ?

Layer

Détermine l'ordre des panneaux. Dans la suite du tutorial nous allons créer un village avec derrière une montagne et encore derrière le ciel, 3 images à superposer, chacune aura un numéro de layer.



Bon assez de théorie, passons à un peu de pratique.

Notre objectif est de dessiner une ville style Far West et de faire se promener un cow-boy. Et bien qu'en 2 D nous allons essayer de donner un peu de relief à tout cela.

Pour ce faire nous allons utiliser une astuce de cinéma qui consiste à faire défiler nos 3 vues à des vitesses différentes donnant ainsi une impression de profondeur.

Commençons par positionner notre ciel :

Reprenons notre fichier **anim2d.wdl** et définissons notre panneau de fond. Ce panneau consiste en une image bitmap de 640 x 480 pixels complètement noire. Nous l'avons appelé **fond.pcx.**

Nous la définissons comme suit (à insérer dans nos variables) :

```
bmap fond_map,<fond.pcx>;

panel fond_pan
{
   pos_x = 0;pos_y = 0;
   layer = 0;
   bmap fond_map;
}
```

Et dans notre fonction **game** nous tapons :

```
fond_pan.visible = on;
```

Si nous exécutons le niveau maintenant nous devrions avoir un écran noir après l'affichage du logo.

Mettons un peu de couleur en créant notre ciel. Le ciel consiste en une image bitmap de 640x 120 pixels appelée **ciel.pcx**.

Nous la définissons comme suit (à insérer dans nos variables) :

```
bmap ciel_map,<ciel.pcx>;

panel ciel_pan
{
   pos_x = 0;pos_y = 0;
   layer = 1;
   bmap ciel_map;
   flags = refresh;
}
```

Et à la fin de notre fonction **game** nous tapons :

```
ciel_pan.visible = on;
```

Si nous exécutons le niveau maintenant nous devrions avoir un beau ciel bleu.

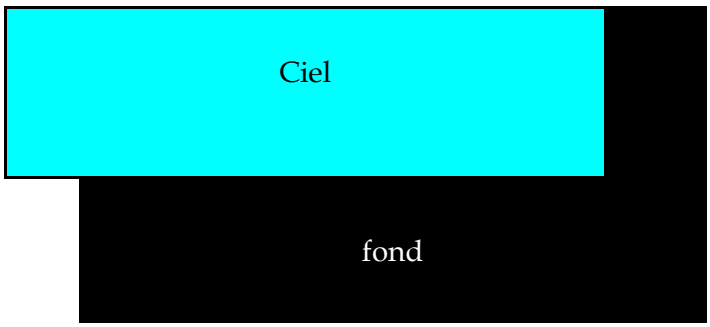
L'idéal serait d'animer ce ciel, qu'en pensez-vous ?

Pour ce faire nous allons utiliser la méthode suivante :

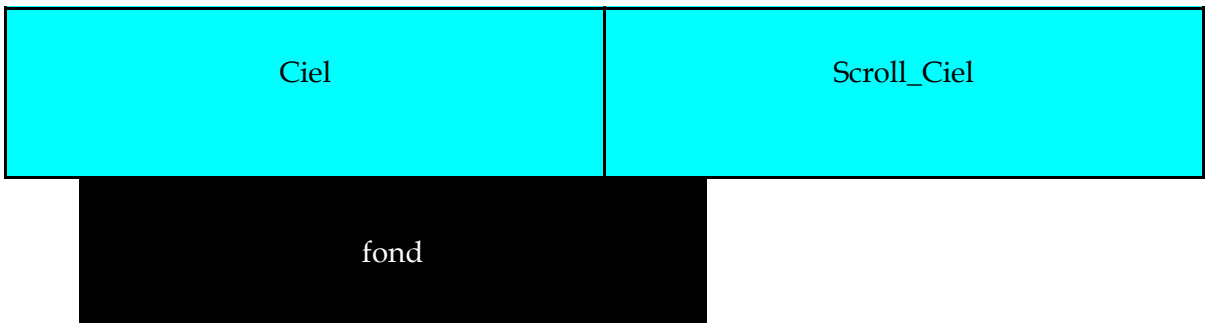
1 – nous définissons un **scroll_ciel_pan** identique à **ciel_pan**

```
panel scroll_ciel_pan
{
   pos_x = 0;pos_y = 0;
   layer = 1;
   bmap ciel_map;
   flags = refresh;
}
```

2 - Nous affichons notre ciel à la position correspondant à la valeur du déplacement que nous souhaitons faire.



3 - Nous affichons notre **`scroll_ciel`** à (640 - la position) correspondant à la valeur du déplacement que nous souhaitons faire.



Ce qui donne ceci :

```
var ciel_pos =0;
```

... au début de nos variables. Puis à la fin de game nous tapons :

```
ciel_pan.visible = on;
scroll_ciel_pan.visible = on;
while(1)
{
     ciel_pan.pos_x = - ciel_pos;
     scroll_ciel_pan.pos_x = 640 - ciel_pos;
     avance_ciel();
     waitt (1);
}
```

et nous tapons notre fonction **`avance_ciel`** :

```
function avance_ciel()
{
      ciel_pos += 1;
      if (ciel_pos >= 640){ciel_pos = 0;}
}
```

Affichons à présent notre montagne. La montagne consiste en une image bitmap de 640x 200 pixels appelée **`mont.pcx`**.

Nous la définissons comme suit (à insérer dans nos variables) :

```
bmap mont_map,<mont.pcx>;

panel mont_pan
{
   pos_x = 0;pos_y = 35; // y = 35 c'est à dire que la montagne est plus basse que le ciel
      layer = 2;
   bmap mont_map;
   flags = overlay,refresh; //overlay pour rendre le noir transparent
}
```

Et dans notre fonction **game**, avant **while(1)** nous tapons :

```
mont_pan.visible = on;
```

Si nous exécutons le niveau maintenant nous devrions avoir un paysage de montagne sur un fond de ciel bleu.

C'est pas mal, qu'en pensez-vous ?

Préparons le scrolling de la montagne, comme pour le ciel en ajoutant les instructions suivantes :

Au début de variables

```
var mont_pos = 0;
```

puis en dessous de la définition du panel **mont_pan:**

```
panel scroll_mont_pan
{
   pos_x = 0;pos_y = 35; // y = 35 c'est à dire que la montagne est plus basse que le ciel
      layer = 2;
   bmap mont_map;
   flags = overlay,refresh; //overlay pour rendre le noir transparent
}
```

et dans **game():**

```
scroll_ mont_pan.visible = on;
```

puis dans notre boucle **while(1)** (ajouter ce qui est en rouge)

```
   while(1)
   {

      ciel_pan.pos_x = - ciel_pos;
      scroll_ciel_pan.pos_x = 640 - ciel_pos;
      mont_pan.pos_x = - mont_pos;
      scroll_mont_pan.pos_x = 640 - mont_pos;
      avance_ciel();
      deplace();
      wait (1);
   }
```

puis nous tapons notre nouvelle fonction **deplace:**

```
function deplace()
{
   if (key_cur == 1) //droite
   {
      ciel_pos += 1;
      if (ciel_pos >= 640){ciel_pos = 0;}
      mont_pos  += 3;
      if (mont_pos >= 640){mont_pos = 0;}

   }
   if (key_cul == 1) //gauche
   {
      ciel_pos -= 1;
      if (ciel_pos <= 0){ciel_pos = 640;}
      mont_pos  -= 3;
      if (mont_pos <= 0){mont_pos = 640;}

   }
}
```

Nous sauvegardons et exécutons notre niveau. Déplacez-vous avec les flèches de direction. Plutôt sympa, non ?

Affichons à présent la partie gauche de la ville, nous utilisons pour cela **`ville.pcx`**, le reste est maintenant de la routine pour vous :

```
bmap ville_map = <ville.pcx>;
var ville_pos = 0;

panel ville_pan
{
   pos_x = 0;pos_y = 85; // y = 85 c'est à dire que la ville est encore plus basse
   layer = 3;
   bmap = ville_map;
   flags = overlay,refresh; //overlay pour rendre le noir transparent
}
```

Et dans notre fonction **`game`**, avant **`while(1)`** nous tapons :

```
ville_pan.visible = on;
```

Puis dans notre boucle **`while(1)`** nous ajoutons :

```
      ville_pan.pos_x = - ville_pos;
```

Pour le déplacement de notre ville nous n'avons pas à faire de scrolling circulaire car nous devons nous arrêter à chaque extrémité. Donc un seul panneau nous suffit. Nous devons par contre arrêter le scrolling du ciel et de la montagne lorsque nous sommes à chacune des extrémités.

Nous ajoutons les instructions suivantes à notre fonction **`deplace`** (les ajouts sont en rouge)

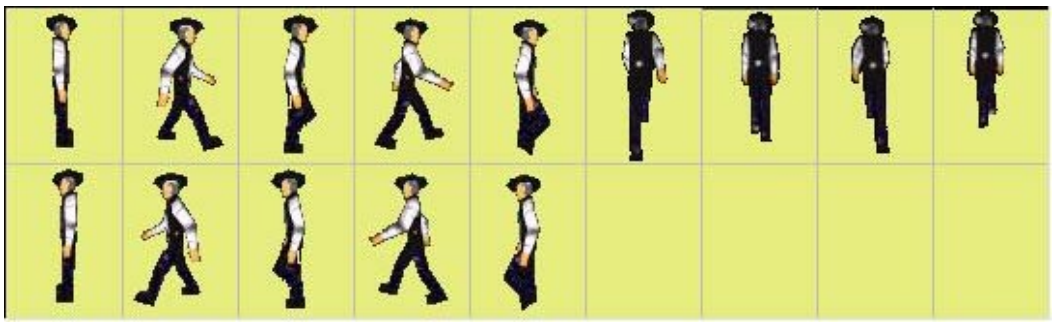
```
   if ((key_cur == 1) && (ville_pos < 1600)) //droite
   {
      ville_pos += 6;
- -  --  - - -  -

   if ((key_cul == 1) && (ville_pos >134)) //gauche
   {
      ville_pos -= 6;
```

Nous pouvons exécuter notre niveau et nous promener dans la ville dans un sens ou dans l'autre.

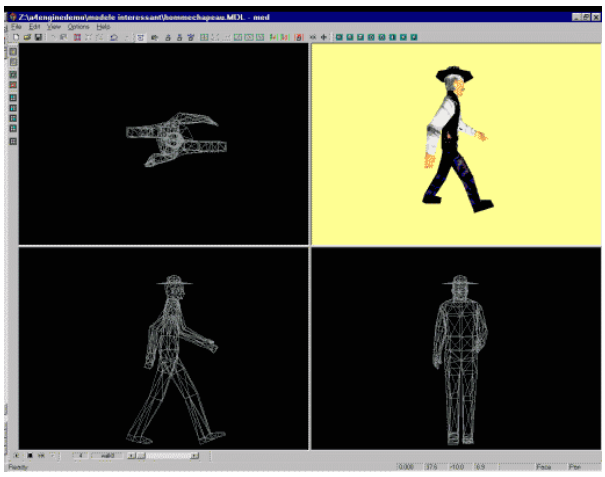
L'affichage du cow-boy

Reprenons un peu notre théorie, l'affichage du cow-boy demandant quelques explications. En effet le cow-boy est une image animée. Voici l'image bitmap que nous allons utiliser :



J'ai mis un fond jaune pour une meilleure lisibilité mais je vous rappelle que ce fond est noir (0,0,0) pour l'utilisation de l'overlay.

La première question que vous pourriez vous poser c'est comment faire un personnage animé en 3D. La réponse est très simple, en utlisant MED et en faisant des captures d'écrans. Voici par exemple la première position de marche :



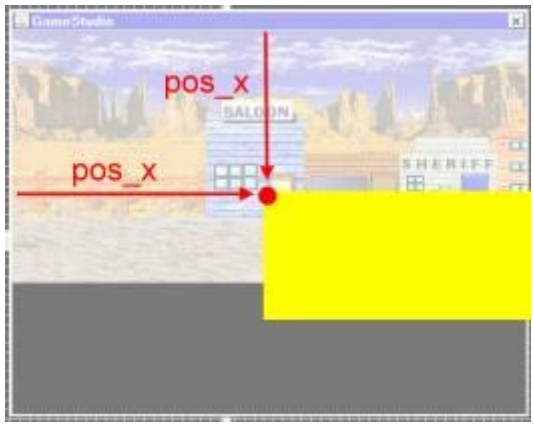
Après une savante découpe, et mise à l'échelle on obtient l'image finale.

Pour notre affichage nous allons utiliser le sous élément window de l'instruction panel.

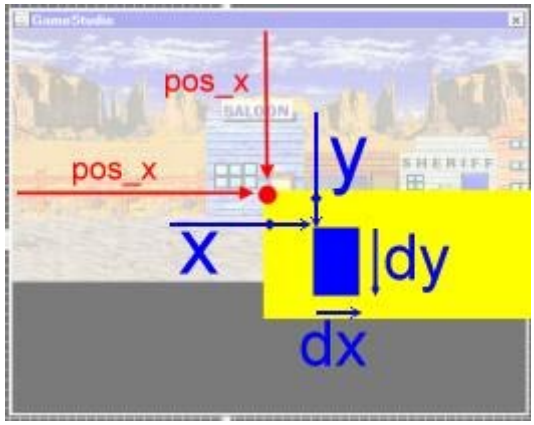
L'instruction s'écrit :

```
window = x,y,dx,dy,bmap,varx,vary;
```

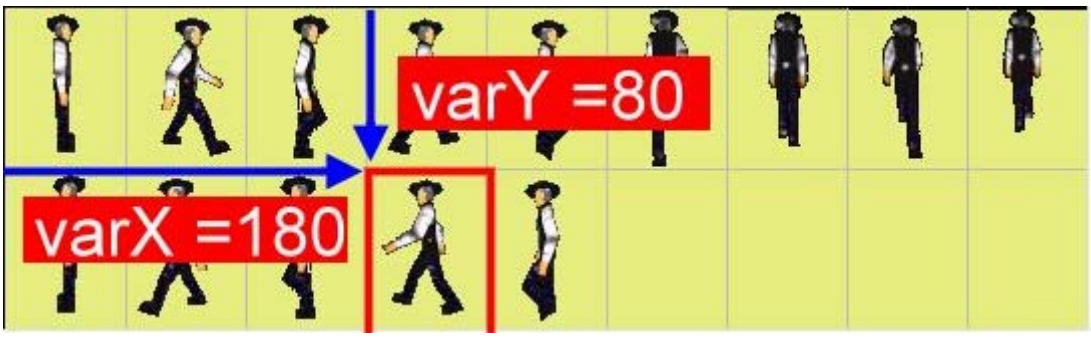
Tout d'abord positionnement de notre panneau (jaune ici) comme ceci :



Ensuite nous positionnons une fenêtre (window) (bleue ici) à la distance x, y du point zéro de notre panneau jaune et d'une largeur de dx et hauteur de dy comme ceci :



Ensuite nous définissons notre image bitmap puis les coordonnées d'un point qui va lui permettre de découper une image de la taille de notre fenêtre bleue (exemple pour afficher notre 4ème cow-boy du bas) :



Ce que nous écrivons comme suit :

Dans nos variables

```
var cowboy_frame_pos[2]=0,0;
```

La première valeur correspondant à notre **`varX`** et qui aura pour valeur 0, 60, 120, 180 et 240 pour la marche sur le côté.
La deuxième valeur correspondant à notre **`varY`** et qui aura pour valeur 0 si on marche vers la droite et 80 si on marche vers la gauche.

Puis nous définissons donc notre panneau comme suit :

```
bmap cowboy _map,<cowboy.pcx>;

panel cowboy _pan
{
   pos_x = 310;pos_y = 200;
      layer = 7;
   window = 0,0,60,80,cowboy _map,cowboy_frame_pos.x,cowboy_frame_pos.y;
         flags = d3d,overlay,refresh;
}
```

Nous retrouvons bien entendu nos valeurs **pos_x** et **pos_y**. Ces valeurs correspondent au cow-boy du départ devant la porte du saloon.

Pourquoi layer 7 alors que nous n'en avons défini que 4 (0 à 3) ? Simplement parce qu'on est prévoyant, c'est le propre de tout bon programmeur, et que peut-être plus tard il se passera des choses entre les maisons et le cow-boy.

Ensuite nous trouvons la définition de notre fenêtre 0, 0, 60, 80 c'est à dire que nous souhaitons que son coin supérieur gauche soit le même que celui du panneau qui l'héberge, et sa taille est de 60 pixels en largeur et 80 pixels en hauteur. Puis **cowboy_map** qui est le nom de l'image dans lequel nous allons piocher et pour finir l'origine de l'image à piocher.

Ouf ! Cela méritait bien quelques explications.

Il ne nous reste plus qu'à afficher le panneau, c'est fait en mettant cette instruction dans le début de **game**, avant le **while(1).**

```
cowboy_pan.visible = on;
```

Et à gérer le mouvement du cow-boy par les lignes suivantes (en rouge) :

```
if ((key_cur == 1) && (ville_pos < 1570)) //droite
{
   cowboy_frame_pos.x +=60;cowboy_frame_pos.y = 0;
   if (cowboy_frame_pos.x >= 300){cowboy_frame_pos.x = 60;}
- - - -  --  - - - - -

if ((key_cul == 1) && (ville_pos >128)) //gauche
{
   cowboy_frame_pos.x +=60;cowboy_frame_pos.y = 80;
   if (cowboy_frame_pos.x >= 300){cowboy_frame_pos.x = 60;}
```

On exécute le programme et on se fait plaisir en se promenant.

Mais le plaisir est vite gâché par tous ces petits défauts qui nous sautent aux yeux.

Premier défaut, lorsqu'on s'arrête le joueur est en état de marche (jambe en l'air) ce n'est pas génial.

Deuxième amélioration à apporter, ce serait bien de n'avoir à appuyer qu'une fois sur la flèche de direction et faire en sorte que le joueur aille devant la porte suivante. (Vous l'aurez compris la finalité étant d'entrer dans les bâtiments).

Que nous faut-il pour cela :

Une variable direction, un compteur de pas et une variable mouvement. On y va. Dans nos variables on tape :

```
var waiting = 0;
var go_right = 1;
var go_left = 2;
var state = 0;

var look_right = 1;
var look_left = 2;
var look_at = 1;

var compteur_pas = 0;
```

et nous modifions sensiblement notre routine **deplace**

La voici dans son entier :

```
function deplace()
{
   if ((key_cul == 1) && (state == waiting))
   {
      state = go_left;
      compteur_pas = 20;
      if (key_ctrl == 1){ compteur_pas = 40;}
   }

   if ((key_cur == 1) && (state == waiting))
   {
      state = go_right;
      compteur_pas = 20;
      if (key_ctrl == 1){ compteur_pas = 40;}
}


   if ((state == go_right) && (look_at == look_left))
   {
      state = waiting;
      look_at = look_right;
      cowboy_frame_pos.x = 0;cowboy_frame_pos.y = 0;
   }

   if ((state == go_right) && (compteur_pas > 0))
   {
      look_at = look_right;
      if (ville_pos < 1570) //droite
      {
         cowboy_frame_pos.x +=60;cowboy_frame_pos.y = 0;
         if (cowboy_frame_pos.x >= 300){cowboy_frame_pos.x = 60;}
         ville_pos += 6;
         ciel_pos += 1;
         if (ciel_pos >= 640){ciel_pos = 0;}
         mont_pos  += 3;
         if (mont_pos >= 640){mont_pos = 0;}

         compteur_pas -=1;
         if (compteur_pas ==0)
         {
            state = waiting;
            cowboy_frame_pos.x = 0;cowboy_frame_pos.y = 0;
         }
      }
      else
```

```
      {
         compteur_pas = 0;
         state = waiting;
         cowboy_frame_pos.x = 0;cowboy_frame_pos.y = 0;
      }

   }

   if ((state == go_left) && (look_at == look_right))
   {
      state = waiting;
      look_at = look_left;
      cowboy_frame_pos.x = 0;cowboy_frame_pos.y = 80;
   }

   if ((state == go_left) && (compteur_pas > 0))
   {
      look_at = look_left;
      if (ville_pos >128) //gauche
      {
         cowboy_frame_pos.x +=60;cowboy_frame_pos.y = 80;
         if (cowboy_frame_pos.x >= 300){cowboy_frame_pos.x = 60;}
         ville_pos -= 6;
         ciel_pos -= 1;
         if (ciel_pos <= 0){ciel_pos = 640;}
         mont_pos  -= 3;
         if (mont_pos <= 0){mont_pos = 640;}

         compteur_pas -=1;
         if (compteur_pas ==0)
         {
            state = waiting;
            cowboy_frame_pos.x = 0;cowboy_frame_pos.y = 80;
         }
      }
      else
      {
         compteur_pas = 0;
         state = waiting;
         cowboy_frame_pos.x = 0;cowboy_frame_pos.y = 80;
      }
   }
}
```

Vous pouvez exécuter le niveau est vous promener.
Vous constatez que :

- Un changement de direction à l'arrêt le fait simplement changer de direction sans avancer
- L'appui d'une touche de direction fait avancer notre cow-boy d'une porte vers la suivante.

C'est pas mal reconnaissez-le. Mais ça manque un peu d'animation, vous ne trouvez pas ?

Nous allons faire apparaître un cavalier qui traversera la ville de façon aléatoire.
Voici l'image bitmap (le fond est normalement noir et les traits de séparation n'existent pas)

A la question habituelle : 'mais comment faire un cheval animé ?" La réponse est toujours la même,

j'ai utilisé MED. A une différence près, je n'ai pas trouvé de cheval animé il m'a donc fallu créer de toute pièce l'animation. (Comme ce n'est pas l'objet de ce tutorial je n'entre pas dans les détails mais peut-être trouverez-vous l'explication un jour dans un mini tutorial ou plus probablement dans Aum à la rubrique utilisateur.)

Pour l'animation du cheval nous allons bien entendu utiliser le sous élément window comme pour le cow-boy, la différence principale étant que contrairement au cow-boy le cheval va devoir en plus se déplacer donc il nous faudra également déplacer le panneau. Comment s'y prend-on ?

Faisons cela en 2 étapes, animons d'abord le cheval au milieu de l'écran :

Nous créons une variable:

```
var cheval_frame_pos = 0;
```

Puis nous créons notre panneau :

```
bmap cheval_map,<cheval.pcx>;
panel cheval_pan
{
   pos_x = 400;pos_y = 200;
      layer = 8;
   window = 0,0,140,102,cheval_map,cheval_frame_pos.x,0;
         flags = d3d,overlay,refresh;
}
```

`cheval_frame_pos.x` aura les valeurs 0, 140, 280, 420 et 560. A la valeur 700 on remet 0.

Il ne nous reste plus qu'à afficher le panneau, c'est fait en mettant cette instruction dans le début de **`game`**, avant le **`while(1)`**

```
   cheval_pan.visible = on;
```

Et à gérer l'animation du cheval par les lignes suivantes (en rouge) :

```
function deplace()
{
        cheval_frame_pos.x +=140;
        if (cheval_frame_pos.x >= 700){cheval_frame_pos.x = 0;}

   if ((key_cul == 1) && (state == waiting))
```

On exécute et on regarde. N'oubliez pas d'applaudir.

Il ne nous reste plus qu'à déplacer notre cheval et à le faire apparaître de façon aléatoire. Pour nos tests nos remplaçons la fonction aléatoire par l'appui sur une touche (espace par exemple)

```
on_space anim_cheval;

function anim_cheval
{
   cheval = 1;
}
```

et au début de nos variables

```
var cheval = 0;
```

La difficulté (petite je vous rassure) du déplacement du cheval vient du fait que nous nous déplaçons pas par rapport à notre écran mais par rapport à notre ville qui elle-même bouge du fait du scrolling.

Nous ajoutons dans nos variables :

```
var cheval_pos = 0;
```

Et dans notre fonction **game** juste avant l'appel de la fonction **deplace** :

```
cheval_pan.pos_x = cheval_pos - ville_pos;
deplace();
```

Et comme nous sommes impatients nous regardons vite ce qui se passe. (Il ne se passera rien tant que vous n'aurez pas pressé la barre espace). Oh que c'est beau…

Mais quelle frustration, que se passe-t-il vraiment avant et après notre écran ?, Le cheval avance-t-il ? Sommes-nous sûr que tout fonctionne comme on veut. Et lorsqu'on va mettre la fonction aléatoire et que le cheval ne vient pas, est-ce parce qu'il y a un bug, est-ce la fonction aléatoire qui ne donne pas la valeur qu'on attend, ce n'est même plus de la frustration, c'est de l'angoisse métaphysique. Suis-je bon ou ne suis-je pas ?

On dispose bien entendu de la fonction debug mais elle n'est pas appropriée pour répondre à nos angoisses. Non l'idéal serait que nous puissions afficher les variables que nous souhaitons sur notre écran pendant la phase de mise au point. Nous pourrions bien entendu reprendre la fonction **D** et lui faire afficher nos variables mais c'est tellement mieux de le faire soi même (nous sommes la pour apprendre oui ou non ?) Et nous préparerons ainsi la suite du jeu car il y a des choses à écrire.

Qui dit affichage de texte, dit utilisation d'une fonte. Qu'est-ce qu'une fonte ? C'est un fichier qui nous donnera l'image du caractère que nous souhaitons imprimer. Nous souhaitons afficher par exemple un 'A' il apparaîtra à l'écran sous sa représentation graphique :

A ou *A* ou E ou A ou *A* ou ***A*** ou A ou A ou A ou A ou A ou *A* ou ✌ ou ✌ ou B(et oui) …

J'ai mis des points de suspension car la seule limite reste votre imagination.

Si la codification des caractères est normalisée (heureusement sinon ce serait l'anarchie), le 'A' par exemple dans la normalisation ASCII (celle que nous utilisons) a toujours la valeur 64, l'espace qui est le premier caractère dit imprimable a la valeur 32. Le 0 à la valeur 48 etc. leur représentation graphique vous appartient. Vous avez le droit de dire 'lorsque je te demande de m'afficher le caractère 64, dessine-moi une petite voiture.'

Mais avant de démarrer prenons un peu de temps pour réfléchir et posons-nous les bonnes questions ?

- qu'ai-je besoin d'afficher ? Si ce ne sont que des nombres je vais pouvoir faire une fonte qui ne comprendra que 11 caractères (l'espace + les 10 chiffres). Si j'utilise la table ascii 7 bits je vais faire une fonte de 128 caractères et si j'utilise la table ascii étendue je vais faire une fonte de 256 caractères.

- Qu'elle est la taille des caractères que je souhaite avoir ?

- Vais-je utiliser les dessins d'une fonte existante ou vais-je redessiner mes propres caractères?

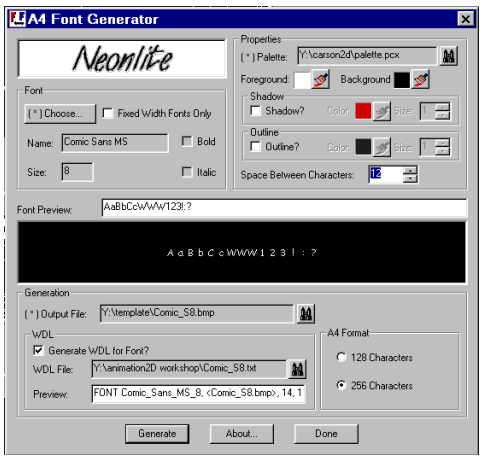
Dans notre cas si nous voulons être international et gérer les spécificités à chaque langue nous devrions utiliser l'ascii étendu c'est à dire une fonte de 256 caractères.

Il est à noter qu'il y aura une différence à fonte identique, entre notre écriture sous windows et la même avec le moteur 3D Gamestudio. En effet windows utilise l'espacement proportionnel (c'est à dire que l'espacement entre caractère est variable selon le caractère, la police etc., un "i" prend moins de place qu'un "w" par exemple. A4/A5 gère les polices comme des polices à espacement fixe.

La preuve la même ligne écrite dans différentes polices :

Police arial corps 11	Abcdefghijkl WAIW	espacement proportionnel
Police agency corps 11	Abcdefghijkl WAIW	espacement proportionnel
Police courrier corps 11	Abcdefghijkl WAIW	Espacement fixe

Ceci étant dit, la communauté 3D GameStudio étant formidable, vous trouverez sur la page des liens un utilitaire qui vous permet de générer votre fonte à partir d'une fonte windows. Voici comment il se présente :



Et voici le résultat :

Il ne nous reste plus qu'à la définir dans notre scénario (on peut recopier le fichier TXT généré par A4 font generator). A taper au début de nos variables

```
font comic, <comic_s8.bmp>,10,15;
```

Le résultat que nous souhaitons est celui-ci :

les coordonnées du texte se mesurent par rapport à l'origine de la fenêtre	
	Les coordonnées des variables se mesurent par rapport à l'origine du panneau qui nous héberge.

Nous allons donc définir notre texte comme suit :

```
string mouchard = "cheval     cheval_pos     ville_pos";
text mouchard_txt
{
   pos_x = 10;pos_y = 350; // distance par rapport au 0,0 de l'écran
   layer = 10;
   font = comic;
   string = mouchard;
}
```

puis nous préparons l'affichage de nos variables :

```
panel affiche_var_pan
{
   pos_x = 0;pos_y = 380; // position de notre panneau par rapport au 0,0 de l'écran
      layer = 11;
   bmap fond_map; //on reprend notre fond noir d'origine
   digits = 35,0,1,comic,1,cheval;
   digits = 140,0,4,comic,1,cheval_pos;
   digits = 270,0,4,comic,1,ville_pan.pos_x;
         flags = overlay,refresh;
}
```

Comment lisons-nous ceci : (prenons en exemple la ligne en bleu)
L'instruction est : **`digits = x,y,len,font,factor,var ;`**

Digits = 35, //affiche à 35 pixels du bord gauche du panneau 'affiche_var' ➔ **X**
0, //affiche à 0 pixel du bord haut du panneau 'affiche_var' ➔ **Y**
1, //nombre de caractère d'affichage (notre variable valant 0 ou 1, 1 caractère suffit ➔ **len**
comic, //utilise la fonte 'comic' ➔ **font**
1, //multiplie le résultat par 1 ➔ **factor**
cheval ; //le contenu de la variable appelée 'cheval' ➔ **var**

factor est utile car digits n'affiche que la partie entière de la variable. Si votre variable fait 0,123 digits affichera 0. Vous mettrez donc factor = 1000 et digits affichera 123.

Astuce : si dans l'image bitmap nous avions par exemple mis un petit rond rouge à la place du 0 et un petit rond vert à la place du 1, l'affichage du contenu de la variable 'cheval' ne donnerait plus 0 ou 1 mais rond rouge ou rond vert.

Il ne nous reste plus qu'à provoquer l'affichage par l'appui sur la touche F12 par exemple (à placer à la fin du scénario):

```
on_f12 espion;

function espion()
{
   if (mouchard_txt.visible == off)
   {
   affiche_var_pan.visible = on;
   mouchard_txt.visible = on;
}
else
   {
   affiche_var_pan.visible = off;
   mouchard_txt.visible = off;
}

}
```

Nous exécutons et en appuyant sur **[F12]** nous faisons apparaître l'affichage et en appuyant sur **[espace]** on voit la variable **`cheval`** passer de **`0`** à **`1`** et la position du cheval qui s'incrémente. Si nous avançons notre cow-boy, la position de la ville se modifie également. Ce qui nous permet de constater que tout va bien.

Nous en profitons pour faire que le cheval ne démarre plus par l'appui sur la barre **[espace]** mais aléatoirement.

Nous modifions le début de notre fonction **`deplace`** comme suit (ligne en rouge) :

```
   if (cheval == 1)
   {
      cheval_frame_pos.x +=140;
      if (cheval_frame_pos.x >= 700){cheval_frame_pos.x = 0;}
      cheval_pos += 12;
      if (cheval_pos > 2185)
      {
         cheval_pos = 0;
         cheval = 0;
      }
   }
   else {cheval = int(random(100));}
```

nous supprimons les lignes suivantes (on peut encore les garder mais il faudra penser à les enlever lorsque votre jeu est terminé !):

```
on_space anim_cheval;

function anim_cheval
{
   cheval = 1;
}
```

et nous modifions la ligne suivante (le bleu est remplacé par le rouge) :

avant :

```
digits = 35,0,1,comic,1,cheval;
```

après

```
digits = 35,0,2,comic,1,cheval;
```

En effet nous souhaitons l'affichage sur 2 caractères pour voir le résultat de **int**(random(100)).

Les sons

Mais que serait notre jeu sans le moindre bruitage, le monde du silence ?
Nous allons commencer par le plus facile et le plus évident, les pas du cow-boy lorsqu'il marche.

Nous créons 2 variables pour le son que nous insérons dans nos variables :

```
sound tap, <tap.wav>;
var taphandle = 0;
```

Puis nous jouons le son lorsque le cow-boy se déplace et nous l'arrêtons dés que le cow-boy s'arrête.

A copier dans nos routines de déplacement (lignes en rouge uniquement) :

```
if ((state == go_right) && (compteur_pas > 0))
{
   if (taphandle ==0) //le son n'est pas joué
   {
      play_loop (tap,20);
      taphandle = result;
   }

    -   - - - - - - - - - - - - - -

   else
   {
      compteur_pas = 0;
      state = waiting;
      cowboy_frame_pos.x = 0;cowboy_frame_pos.y = 0;
   }
   if (state == waiting)
   {
      stop_sound (taphandle);
      taphandle = 0;
   }

    -   - - - - - - - - - - - - - -

if ((state == go_left) && (compteur_pas > 0))
{
   if (taphandle ==0) //le son n'est pas joué
   {
      play_loop (tap,20);
      taphandle = result;
```

```
      }
       -   - - - - - - - - - - - - - -

      else
      {
         compteur_pas = 0;
         state = waiting;
         cowboy_frame_pos.x = 0;cowboy_frame_pos.y = 80;
      }
      if (state == waiting)
      {
         stop_sound (taphandle);
         taphandle = 0;
      }
- - - - - - - - - - - - - -
```

Ca n'était pas le plus dur, même si ce n'est pas génial, je vous laisse chercher des pas plus appropriés si vous le souhaitez.

Là où ça devient intéressant c'est avec le pas du cheval. Il est évident qu'en fonction de l'éloignement du cheval par rapport au cow-boy, le volume des pas du cheval ne sera pas le même.

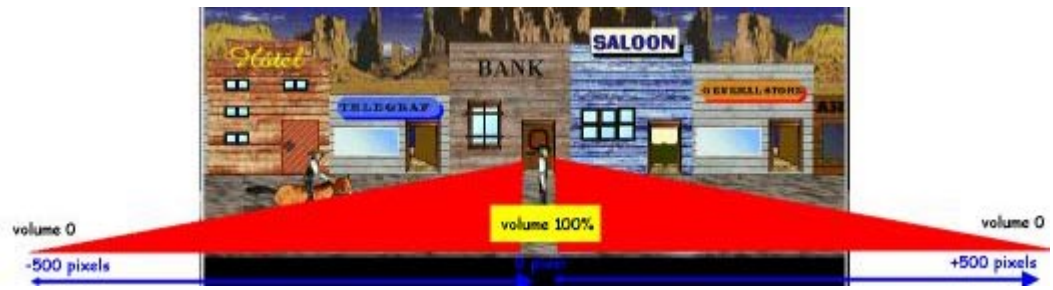
Lorsque l'on regarde attentivement les instructions relatives aux sons, la seule qui propose une variation de volume est :

```
tune_sound(handle,varvol,varfreq);
```

Nous allons bien entendu jouer avec la variable **`varvol`**.

Ce que nous voulons : plus le cheval s'approche de moi, plus le son est fort, plus il s'éloigne, plus le son est faible.

J'ai fixé une règle complètement arbitraire pour dire qu'au-delà de 500 pixels d'écart entre le cheval et le cow-boy, on n'entend pas le son et qu'à partir de 500 pixels jusqu'à 0 pixels le volume du son va en augmentant. Ce que l'on peut représenter comme suit :



Etant bien entendu que ce n'est pas directement la position du cheval par rapport à nous qui nous intéresse, mais la distance du cheval par rapport à la ville par rapport à nous. En effet bien que nous soyons toujours au milieu de l'écran, notre position par rapport à la ville est variable. Ceci étant dit, cette position est déjà calculée puisqu'elle nous sert à l'affichage du panneau qui héberge notre cheval, elle a pour nom : **`cheval_pan.pos_x`**. Nous la soustrayons à notre position par rapport à l'écran (le milieu = 320), le résultat donne une valeur comprise entre 500 et 0 (en plus ou en moins), nous divisons donc notre résultat par 5 pour avoir une valeur comprise entre 0 et 100 qui sont justement les 2 bornes du réglage de volume. Nous prenons bien entendu la valeur absolue pour faire abstraction du signe.

Le petit problème étant que nous avons un résultat inversé, un volume à 0 lorsque le cheval est sur

nous et un volume à 100 lorsque le cheval est éloigné. Nous soustrayons donc ce résultat de 100 et tout rentre dans l'ordre.

En algèbre informatique cela s'écrit :

```
volume = 100-(abs(320-cheval_pan.pos_x)/5);
```

Faites-moi confiance, je vous dis que ça fonctionne.

Il ne nous reste plus qu'à saisir tout cela dans notre scénario. Nous créons les variables suivantes :

```
sound pascheval, <pascheval.wav>;
var paschevalhandle = 0;
var volume = 0;
```

puis dans notre routine de déplacement (lignes en rouge) :

```
function deplace()
{
   if (cheval == 1)
   {
      if (paschevalhandle ==0) //le son n'est pas joué
      {
         play_loop (pascheval,5);
         paschevalhandle = result;
      }
      else
      {
         volume = 100-(abs(320-cheval_pan.pos_x)/5);
         tune_sound (paschevalhandle,volume,0);
      }

      cheval_frame_pos.x +=140;
      if (cheval_frame_pos.x >= 700)
      {
         cheval_frame_pos.x = 0;
      }
      cheval_pos += 12;
      if (cheval_pos > 2185)
      {
         cheval_pos = 0;
         cheval = 0;
      }
   }
   else
   {
      cheval = int(random(100));
      stop_sound(paschevalhandle);
      paschevalhandle = 0;
   }
```

On exécute le niveau et on se fait un petit plaisir.

C'est pas mal mais je suis quand même un peu déçu. J'ai le casque sur les oreilles, ma carte sonore permet des effets stéréo, ce serait bien si le son était à gauche quand le cheval est à gauche et le son à droite quand le cheval est à droite. Qu'en pensez-vous ?

J'ai beau relire les paramètres de l'instruction, rien ne permet de faire cette distinction. Peut-être n'ai-je pas pris la bonne instruction. Poursuivons notre lecture du manuel WDL.

Est-ce que l'instruction **play_entsound** (my, sound,var)' qui joue un son 3D ne serait pas plus appropriée ? Oui mais les explications nous parlent d'entité **my**, de **camera**, nous n'avons rien de cela nous sommes en jeu 2D.

Détrompez-vous, le moteur A4/A5 est tellement fantastique que même en 2D, nous avons une camera et nous pouvons créer des entités. Splendide, n'est-ce pas ?

Nous allons donc créer une entité bidon que nous attacherons à notre cheval et nous allons positionner la caméra à l'emplacement de notre joueur.

On y va, tapez ces lignes :
Dans nos variables

```
synonym bidon {type entity;}
var bidon_pos[3] =0,200,0;
```

Dans notre fonction **main** tapez la ligne en rouge:

```
load_status();
   create <bidon.pcx>,bidon_pos,f_bidon;
   game();
}

function f_bidon
{
   bidon = me;
   while (bidon == null){wait 1;}
}
```

Dans notre boucle while de **game** nous ajoutons les lignes en rouge :

```
   cheval_pan.visible = on;
   camera.x = 0;
   camera.y = 0;
   camera.z = 0;

   while(1)
   {

      ciel_pan.pos_x = - ciel_pos;
      scroll_ciel_pan.pos_x = 640 - ciel_pos;
      mont_pan.pos_x = - mont_pos;
      scroll_mont_pan.pos_x = 640 - mont_pos;
      ville_pan.pos_x = - ville_pos;
      //avance_ciel();
      bidon.x = cheval_pan.pos_x+60;
```

et dans notre fonction **deplace** vous remplacez les lignes en bleu par les lignes en rouge:

```
function deplace()
{
   if (cheval == 1)
   {
      if (paschevalhandle ==0) //le son n'est pas joué
      {
         play_loop (pascheval,5);
         paschevalhandle = result;
      }
      else
```

```
      {
         volume = 100-(abs(320-cheval_pan.pos_x)/5);
         tune_sound (paschevalhandle,volume,0);
      }


   if (cheval == 1)
   {
      if (paschevalhandle ==0) //le son n'est pas joué
      {
         play_entsound (bidon,paschevalhandle,200);
         paschevalhandle = result;
      }
      else {if (snd_playing(paschevalhandle)==0){paschevalhandle =0;}}
```

On exécute notre niveau et on se promène. C'est tout de suite plus réaliste, vous ne trouvez-pas ?

Vite fait à présent que vous avez tout compris, nous allons faire hennir le cheval, aléatoirement lorsqu'il se trouve près du joueur.

Nous saisissons les lignes en rouge :

```
sound paschevalhandle, <paschevalhandle.wav>;
var paschevalhandle = 0;
sound henni, <henissem.wav>;
var hennihandle = 0;
- - - - - - - - - - - -

   if (cheval == 1)
   {
      //henni si moins de 100 pixels du joeur et 1 fois sur 3
      if ((abs(320-cheval_pan.pos_x) < 100) && (random(3) <1))
      {
         if (hennihandle ==0) //le son n'est pas joué
         {
            play_entsound (bidon,henni,100);
            hennihandle = result;
         }
         else {if (snd_playing(hennihandle)==0){hennihandle =0;}}

      }

      if (paschevalhandle ==0) //le son n'est pas joué
```

Conclusion

Et oui c'est déjà terminé, ce didacticiel a uniquement pour vocation de vous donner vos premières bases pour créer vos propres animations 2D.

Pour la petite histoire, Carson City est un jeu qui a tout d'abord été développé sur Spectrum sinclair puis sur Amstrad en assembleur.

J'avais ensuite repris le développement du projet avec C++ et Direcdraw mais l'arrivée de 3D Gamestudio a chamboulé mes plans.

De ce fait je pense faire une version 3D, mais peut-être que la version 2D avec 3D Gamestudio, qui est quand même plus simple à mettre en œuvre que C++ verra également le jour.

Pour l'instant j'espère que vous avez eu du plaisir à faire un petit bout de chemin avec moi dans cet univers passionnant qu'est la programmation des jeux et que j'ai pu vous faire partager un peu de mon savoir et beaucoup de ma passion.

Visitez mon site : http://alainbrgeon.free.fr

Si vous ne souhaitez pas réécrire entièrement le scénario, vous trouverez une version dans votre répertoire nommée anim2d.al1. Il vous suffit de renommer ce fichier en anim2d.wdl ou de l'ouvrir et de faire du copier / coller avec votre scénario.