13

Améliorer le rendu avec le High Level Shader Language

Certains diront que les effets graphiques sont l'ingrédient indispensable à la réalisation d'un bon jeu, d'autres répliqueront qu'il ne s'agit que de fioritures visuelles qui exigent du matériel graphique de plus en plus puissant. Cependant, impossible de faire l'impasse sur des termes comme glow, blur, ou bloom. En effet, aujourd'hui, toutes les grandes productions en usent, et parfois en abusent.

Ce chapitre présente les shaders ainsi que le langage que vous utiliserez pour les programmer, à savoir le HLSL. Vous découvrirez aussi plusieurs exemples de shaders et comment les utiliser dans un projet XNA.

La première partie de ce chapitre explique ce que sont les shaders et présente le HLSL. Vous découvrirez ensuite la syntaxe du HLSL.

Les shaders et XNA

Un shader est un programme directement exécutable par la carte graphique. Il permet de transférer la gestion des lumières, ombres, textures, etc., au processeur graphique et de soulager de cette charge de travail le processeur de l'ordinateur. Ainsi, à partir du même contenu de base, mais en utilisant divers shaders, vous pouvez obtenir des rendus complètement différents.

Vertex shaders et pixel shaders

On distingue deux types de shaders : les vertex shaders et les pixels shaders.

Les vertex shaders sont exécutés sur tous les vertices d'un objet. Si vous dessinez un simple triangle à l'écran, le vertex shader sera exécuté pour les trois vertices qui composent ce triangle. Mais les vertex shaders peuvent également être exécutés sur des sprites en 2D : en effet, ceux-ci sont constitués de 4 vertices (un dans chaque coin du sprite).

Quant aux pixels shaders, ils sont exécutés sur tous les pixels visibles à l'écran de l'objet que vous dessinez, qu'il s'agisse d'un objet en 3D ou d'un sprite en 2D.

Chronologiquement, le processus est le suivant :

- 1. Le GPU (processeur graphique) reçoit des paramètres et des vertices (leur position, couleur, texture, etc., tout dépend du choix que vous avez fait).
- 2. Le vertex shader est d'abord exécuté. En sortie, vous disposez des vertices modifiés (ou non, tout dépend du shader).
- Ces données passent ensuite à l'étape de rastérisation. Au cours de cette étape, les données vectorielles sont converties en données composées de pixels pouvant être affichés à l'écran.
- 4. Le résultat de la rastérisation est ensuite envoyé au pixel shader. Après l'exécution de ce programme, le GPU retourne la couleur qui devra être affichée.

Comme pour un processeur classique, le processeur graphique ne comprend que des instructions de bas niveau. Ainsi, pour faciliter le travail des développeurs, des langages de plus haut niveau (c'est-à-dire, plus intelligibles) à la syntaxe très proche des langages tels que le C ont été créés. Du côté d'OpenGL, le langage standardisé est le GLSL. Le fabricant de cartes graphiques NVidia a, quant à lui, créé le langage Cg. Enfin, de son côté, Microsoft a créé le HLSL (*High Level Shader Language*). C'est ce dernier langage qui est utilisé par l'API DirectX, et donc par XNA.

Définition

OpenGL est une API multi-plate-forme d'affichage en trois dimensions développée par Silicon Graphics.

Dans XNA, pour afficher quelque chose à l'écran vous devez passer obligatoirement par HLSL. Cependant, pour vous faciliter les choses (c'est le mot d'ordre de XNA après tout !) et vous éviter de devoir vous attaquer directement au HLSL, les développeurs ont créé les classes SpriteBatch et BasicEffect. En arrière-plan, ces classes travaillent directement avec des shaders. Vous pouvez d'ailleurs retrouver le code source de ces shaders à cette adresse : http://creators.xna.com/fr-FR/education/catalog/.

Ajouter un fichier d'effet dans XNA

Les fichiers d'effets écrits en HLSL sont automatiquement gérés par le Content Manager de XNA. Exactement comme vous ajoutez n'importe quel type de fichier (une police de caractères par exemple), vous pouvez ajouter un fichier d'effet à XNA (figure 13-1).

Add New Item - Content		? ×
<u>C</u> ategories:	Templates:	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
⊡ Visual C#	Visual Studio installed templates Image: Bitmap File Image: XML File Image: File Image: Sprite Font My Templates Image: Search Online Templates	
A file describing a custom graphics rende	ring effect	
Name: Effect1.fx		
		<u>A</u> dd Cancel

Figure 13-1

Ajout d'un fichier .fx au projet

Vous avez ensuite toute latitude pour éditer le fichier .fx directement dans Visual Studio, avec un bloc-notes ou bien, si vous désirez des fonctionnalités plus avancées, notamment pour déboguer un effet, via FX Composer de NVidia (figure 13-2).

FX Composer

Téléchargez FX Composer à l'adresse suivante :

- http://developer.nvidia.com/object/fx_composer_home.html/
- Puis, installez le programme comme une application classique.



Figure 13-2

FX Composer, l'éditeur de shaders de NVidia

Syntaxe du langage HLSL

Vous allez maintenant découvrir la syntaxe de base du langage HLSL. Celle-ci étant très proche des langages C/C++/C#, vous l'assimilerez très vite.

Les variables HLSL

Comme la plupart des langages de programmation, le HLSL permet de déclarer des variables de différents types. Vous pouvez ainsi utiliser des variables de type int, float, bool, etc., et même définir vos propres types grâce aux structures.

Définir la matrice

Pour définir les matrices, deux manières différentes s'offrent à vous en HLSL :

• La première manière consiste à utiliser le mot-clé floatLxC, où L est le nombre de lignes et C, le nombre de colonnes. La ligne de code suivante sert donc à déclarer une matrice de 4 colonnes par 4 lignes.

```
float4x4 myMatrix;
```

• La seconde repose sur le mot-clé matrix auquel vous ajoutez le type des données et les dimensions de la matrice :

```
matrix<float,4,4> myMatrix;
```

Accéder au contenu de la matrice

Voici les deux manières d'accéder au contenu de la matrice :

• La première consiste à considérer la matrice comme un tableau de tableaux :

```
float element = myMatrix[0][0];
```

• La deuxième repose sur l'utilisation de la notation pointée « . », comme vous le faites en C# pour accéder aux membres d'une structure ou d'une classe :

```
float element = myMatrix._m23;
```

À la différence du C#, il n'existe pas de types Vector2, Vector3, etc. Vous utiliserez donc les types vector ou floatX, où X est le nombre de composantes utilisées. Ainsi, vous stockerez la position d'un vertex dans une variable de type vector ou float3.

Accéder aux composantes d'un vecteur

L'accès aux différentes composantes d'un vecteur se fait de deux manières :

- La première consiste à utiliser les crochets [], comme vous le feriez pour un tableau en C#.
- float value = myVector[0];
- La deuxième consiste à utiliser la notation pointée « . ». Les composantes peuvent être identifiées par deux noms différents. La première série de noms est rgba et la seconde, xyzw.

Par exemple, si vous avez un vecteur color, vous pouvez accéder à son niveau de vert des deux manières suivantes :

```
float green = color.g;
float green = color.y;
```

Le *swizzling* sert à accéder à plusieurs composantes d'un vecteur en même temps. Dans les deux exemples suivants (qui utilisent les deux méthodes d'accès aux composantes), vous accédez aux composantes rouge et verte d'une couleur :

```
float2 redGreen = { color[0],color[1]};
float2 redGreen = color.rg;
```

Attention à l'erreur !

L'exemple de code suivant est invalide : vous ne pouvez pas mélanger les noms des deux groupes de noms de composantes.

```
float2 redGreen = color.xg;
```

Les structures de contrôle

Les structures de contrôle vous permettont d'effectuer des traitements avancés pour aboutir à des effets encore plus réussis. Vous retrouvez en HLSL une grande partie des structures de contrôle que vous connaissiez déjà en C# et qui s'utilisent de la même manière, à savoir :

- If (si condition vérifiée, alors);
- While (tant que condition vérifiée, faire);
- Do (faire tant que condition vérifiée);
- For (pour X de I a J, faire).

Les fonctions fournies pas le langage

Le langage HLSL fournit une très longue liste de fonctions faciles à utiliser dans les shaders. Le tableau 13-1 en cite quelques-unes. La liste complète se trouve dans la documentation de DirectX, à laquelle vous accéderez via la bibliothèque MSDN (voir annexe B).

Fonction	Description
Cos(x)	Cosinus de x.
Sin(x)	Sinus de x.
Mul(a,b)	Multiplication de la matrice a avec la matrice b.
Pow(x,y)	Retourne x à la puissance y.

Tableau 13-1 Exemples de fonctions fournies pas le langage

Sémantiques et structures pour formats d'entrée et de sortie

Les sémantiques servent à lier les entrées et sorties d'une fonction. Par exemple, elles servent à lier la sortie de l'application XNA à l'entrée du vertex shader. Dans le processus de rastérisation, d'autres sémantiques lient la sortie du vertex shader avec l'entrée. Pour finir cette courte présentation, signalons également que le pixel shader reçoit et renvoie lui aussi des sémantiques.

Vous déclarez vos propres formats d'entrée et de sortie en utilisant les structures. L'exemple ci-dessous définit le format d'entrée pour un vertex shader.

```
struct VertexInput
{
    float4 Position : POSITION;
    float2 TexCoord : TEXCOORDO;
}:
```

Dans la structure précédente, vous constatez que deux sémantiques sont utilisées : une pour la position du vertex et l'autre pour les coordonnées de texture qui lui sont associées. La liste complète des sémantiques disponibles se trouve dans la documentation de DirectX, le tableau 13-2 présente un court extrait de cette liste.

Sémantique	Description
COLOR	Couleur diffuse ou spéculaire.
NORMAL	Normale au vertex.
POSITION	Position du vertex.
TEXCOORD	Coordonnées de texture associées au vertex.

Tableau 13-2 Sémantiques d'entrée pour vertex shader

Procédez de la même manière pour les données en sortie du vertex shader (VertexOutput), les données en entrée du pixel shader (PixelInput) et celles en sortie du pixel shader (PixelOutput). Attention, la liste des sémantiques n'est pas la même pour chacune de ces étapes.

Incompatibilité de versions

Certaines sémantiques ne sont pas toujours valables ou ne s'utilisent pas toujours de la même façon selon la version des vertex shaders ou pixel shaders. Vous devrez donc faire attention à la version des shaders que vous utiliserez lors de la compilation.

Écrire un vertex shader

Vous savez à présent tout ce qu'il faut savoir pour écrire un premier vertex shader. Il s'agit d'une simple fonction qui doit retourner un objet de type VertexOutput, qui est une structure que vous avez normalement déclarée.

```
VertexOutput vertexShader(VertexInput input)
{
    VertexOutput output;
    WorldViewProjection = mul(mul(World, View), Projection);
    output.Position = mul(Pos, WorldViewProjection);
    output.TexCoord = input.TexCoord;
    return(output);
}
```

Ce premier vertex shader multiplie les différentes matrices (World, View et Projection) pour transformer la position du vertex, puis recopier les coordonnées de textures.

Écrire un pixel shader

Le pixel shader est une fonction qui retourne une couleur, c'est-à-dire un vecteur à 4 dimensions.

```
float4 pixelShader(PixelInput input) : COLOR
{
    return tex2D(TextureSampler, input.TexCoord);
}
```

Le pixel shader ci-dessus retourne la couleur de la texture aux coordonnées de texture de l'objet PixelInput.

Finaliser un effet : les techniques et les passes

Comme nous l'avons au chapitre précédent, un fichier d'effets peut contenir plusieurs techniques. Une technique n'est rien de plus qu'un nom et un conteneur de passes. Une passe définit quel vertex shader et quel pixel shader doivent être utilisés. C'est aussi ici que vous choisissez la version des shaders (dans le cas présent, version 1 pour les deux).

```
technique Default
{
    pass P0
    {
        VertexShader = compile vs_1_1 vertexShader();
        PixelShader = compile ps_1_1 pixelShader();
    }
}
```

Vous savez à présent ce qui se cache derrière les termes de vertex shader ou pixel shader. Vous connaissez également la syntaxe du langage HLSL. Bref, vous avez toutes les clés en main pour écrire vos premiers fichiers d'effet.

Créer le fichier d'effet

En reprenant les différents éléments de base d'un fichier d'effets que nous avons vus dans la première partie de ce chapitre, vous reconstituez le fichier suivant. Ce premier fichier d'effet affiche le modèle à l'écran, ni plus, ni moins.

Fichier d'effet basique

```
float4x4 World : WORLD;
float4x4 View;
float4x4 Projection;
float4x4 WorldViewProjection : WORLDVIEWPROJECTION;
texture Texture;
```

```
sampler TextureSampler = sampler_state
{
   texture = <Texture>;
   magfilter = LINEAR;
   minfilter = LINEAR;
   mipfilter = LINEAR;
   AddressU = mirror:
   AddressV = mirror:
}:
struct VertexInput
{
   float4 Position : POSITION;
   float2 TexCoord : TEXCOORDO:
};
struct VertexOutput
{
   float4 Position : POSITION;
   float2 TexCoord : TEXCOORDO;
};
VertexOutput vertexShader(VertexInput input)
{
   VertexOutput output:
   WorldViewProjection = mul(mul(World, View), Projection);
   output.Position = mul(input.Position, WorldViewProjection);
   output.TexCoord = input.TexCoord;
   return( output );
}
struct PixelInput
{
   float2 TexCoord : TEXCOORDO;
}:
float4 pixelShader(PixelInput input) : COLOR
{
   return tex2D(TextureSampler, input.TexCoord);
}
technique Default
{
   pass PO
    {
     VertexShader = compile vs_1_1 vertexShader();
      PixelShader = compile ps_1_1 pixelShader();
    }
}
```

Voyons comment complexifier les choses... Dans un projet XNA, reprenez la classe Cube écrite au chapitre précédent. Modifiez la classe de manière à ne plus utiliser la classe BasicEffect, mais la classe Effect :

```
Effect effect;
```

Chargez ensuite l'effet grâce au Content Manager, comme si vous chargiez une texture, un modèle, etc.

```
effect = Content.Load<Effect>("FirstEffect");
```

Vous accédez aux différentes variables globales de l'effet via la propriété Parameters. Les variables sont ensuite indexées selon leur nom. Renseignez donc les variables World, View, Projection et Texture.

```
effect.Parameters["Projection"].SetValue(projection);
effect.Parameters["View"].SetValue(view);
effect.Parameters["World"].SetValue(Matrix.Identity);
effect.Parameters["Texture"].SetValue(texture);
```

Plusieurs techniques dans le même fichier d'effet

Si vous aviez eu plusieurs techniques dans le fichier, vous auriez pu définir la propriété CurrentTechnique grâce à la propriété Techniques, où les techniques sont indexées par leur nom.

```
effect.CurrentTechnique = effect.Techniques["Default"];
```

Tout est prêt, vous pouvez compiler le projet et le lancer : le cube s'affiche correctement.

Pour compliquer encore un peu les choses, modifions ce premier fichier afin qu'il prenne en charge la lumière ambiante.

Commencez par ajouter une variable globale dans le fichier. Cette variable devra contenir la couleur de la lumière d'ambiance.

```
float4 AmbientColor : COLORO;
```

Vous n'avez plus qu'à appliquer la lumière d'ambiance dans le pixel shader.

```
float4 pixelShader(PixelInput input) : COLOR
{
    return (tex2D(TextureSampler, input.TexCoord) * AmbientColor);
}
```

Le fichier d'effet est maintenant prêt. Repassez dans le fichier cube.cs et définissez la variable AmbientColor.

```
effect.Parameters["AmbientColor"].SetValue(0.5f);
```

Une dernière petite modification pour finir ce premier fichier d'effet : nous allons afficher le cube en mode fil de fer (figure 13-3). Pour cela, il suffit d'ajouter la ligne suivante dans la passe :

```
FillMode = Wireframe;
```





Le cube en mode fil de fer

De la même manière, vous pouvez désactiver le *culling*, ce qui fera bien apparaître toutes les faces du cube (figure 13-4).

```
CullMode = none;
```



Figure 13-4 Le cube en mode fil de fer sans culling

Faire onduler les objets

Le premier fichier d'effets que vous venez de réaliser n'a rien d'extraordinaire. Vous allez maintenant créer un effet plus intéressant que ceux proposés par la classe BasicEffect.

Le but de l'effet suivant est de donner un effet de vague à des vertices : celui-ci oscillera en fonction du temps grâce à la fonction sinus.

- 1. Commencez par ajouter une nouvelle variable globale au fichier qui servira à stocker le temps.
- float Timer : TIME;
- 2. Modifiez ensuite le vertex shader pour qu'il change les coordonnées du vertex, qui est quant à lui passé en paramètre. Pour cela, copiez sa position dans un vecteur temporaire.
- 3. Modifiez la composante x, le calcul utilise la fonction sinus qui prend en paramètre la composante y et le temps courant.
- 4. Dernière chose, passez ce jeu de coordonnées mis à jour au vertex de sortie.

```
VertexOutput vertexShader(VertexInput input)
{
    float4 Pos = float4(input.Position.xyz,1);
    Pos.x += sin(Pos.y + Timer);
    VertexOutput output;
    WorldViewProjection = mul(mul(World, View), Projection);
    output.Position = mul(Pos, WorldViewProjection);
    output.TexCoord = input.TexCoord;
    return( output );
}
```

5. Dans le code C#, à chaque passage dans la méthode Draw(), passez le nombre de secondes écoulées au total.

```
effect.Parameters["Timer"].SetValue((float)gameTime.TotalGameTime.TotalSeconds);
```

Vous pouvez ensuite essayer le projet et admirer le résultat (figure 13-5).





La texture en négatif

À présent, voyons comment modifier le pixel shader de manière à ce que le cube apparaisse en négatif (figure 13-6). Pour cela, vous devez effectuer une simple inversion des couleurs : soustrayez chacune des composantes de la couleur du pixel à 1.

```
float4 pixelShader(PixelInput input) : COLOR
{
    float4 color = 1 - tex2D(TextureSampler, input.TexCoord);
    return( color );
}
```





Jouer avec la netteté d'une texture

Le pixel shader suivant fait varier la netteté de l'image (figure 13-7). Il suffit de modifier la couleur du pixel courant en fonction des pixels voisins.

- 1. Commencez par déclarer une variable globale dans le fichier d'effets.
- float SharpAmount;
- 2. Puis, utilisez le pixel shader suivant.

```
float4 pixelShader(PixelInput input) : COLOR
{
    float4 color = tex2D( TextureSampler, input.TexCoord);
    color += tex2D( TextureSampler, input.TexCoord - 0.0001) * SharpAmount;
    color -= tex2D( TextureSampler, input.TexCoord + 0.0001) * SharpAmount;
    return( color );
}
```



Figure 13-7 *Modification de la netteté du cube*

- Pour apprécier pleinement l'impact de cet effet sur le cube, ajoutez une variable au projet C#.
- float sharpAmount = 0;
- 4. Pour faire varier l'ampleur de l'effet, augmentez la valeur de la variable précédente si le joueur appuie sur la touche PageUp et faites-la diminuer s'il presse la touche PageDown.

```
public void Update(GameTime gameTime)
{
    if (Keyboard.GetState().IsKeyDown(Keys.PageUp))
    {
        sharpAmount += 1;
        effect.Parameters["SharpAmount"].SetValue(sharpAmount);
    }
    else if (Keyboard.GetState().IsKeyDown(Keys.PageDown))
    {
        sharpAmount -= 1;
        effect.Parameters["SharpAmount"].SetValue(sharpAmount);
    }
}
```

Flouter une texture

Le flou (*blur* en anglais) s'obtient presque de la même manière que l'effet précédent : la couleur du pixel est déterminée en fonction de la couleur des pixels voisins (figure 13-8), sauf que cette fois ci vous n'utiliserez pas de seuil de netteté. Pour vous permettre de bien visualiser l'effet, la distance séparant le pixel courant et les pixels voisins sera dynamique.

float Distance;

317

```
float4 pixelShader(PixelInput input) : COLOR
{
    float4 color = tex2D( TextureSampler,
    float2(input.TexCoord.x+Distance, input.TexCoord.y+Distance));
    color += tex2D( TextureSampler,
    float2(input.TexCoord.x+Distance, input.TexCoord.y-Distance));
    color += tex2D( TextureSampler,
    float2(input.TexCoord.x+Distance, input.TexCoord.y-Distance));
    color += tex2D( TextureSampler,
    float2(input.TexCoord.x-Distance, input.TexCoord.y+Distance));
    color += tex2D( TextureSampler,
    float2(input.TexCoord.x-Distance, input.TexCoord.y+Distance));
    color = color / 4;
    return( color );
}
```

Comme pour l'effet précédent, le paramètre dynamique sera modifié si le joueur appuie sur la touche PageUp ou sur la touche PageDown.

```
float distance = 0;
public void Update(GameTime gameTime)
{
    if (Keyboard.GetState().IsKeyDown(Keys.PageUp))
    {
        distance += 0.001f;
        effect.Parameters["Distance"].SetValue(distance);
    }
    else if (Keyboard.GetState().IsKeyDown(Keys.PageDown))
    {
        distance -= 0.001f;
        effect.Parameters["Distance"].SetValue(distance);
    }
}
```

Vous remarquez que sur la figure suivante, la texture appliquée au cube est floutée.

Figure 13-8 Le cube flouté



Modifier les couleurs d'une texture

Les pixels shaders suivants inversent les couleurs d'une texture.

Ici, il renvoie la couleur avec toutes ses composantes dans l'ordre classique.

```
float4 pixelShader(PixelInput input) : COLOR
{
    float4 color = tex2D(TextureSampler, input.TexCoord);
    return (color.rgba);
}
```

Dans le pixel shader suivant, la composante rouge est inversée avec la composante verte.

```
float4 pixelShader(PixelInput input) : COLOR
{
    float4 color = tex2D( TextureSampler, input.TexCoord);
    return( color.grba );
}
```

Il est également possible d'inverser le rouge et le bleu.

```
float4 pixelShader(PixelInput input) : COLOR
{
    float4 color = tex2D( TextureSampler, input.TexCoord);
    return( color.bgra );
}
```

Un effet intéressant consiste à transformer la texture pour que ses couleurs varient uniquement entre le noir et le blanc (on parle également de niveaux de gris). Votre premier réflexe est peut-être de faire la moyenne des trois composantes. Cependant, l'œil humain ne percevant pas de la même manière les trois composantes, vous devez leur appliquer des coefficients. La Commission internationale de l'éclairage conseille d'utiliser les coefficients suivants :

- 0,3 pour le rouge ;
- 0,59 pour le vert ;
- 0,11 pour le bleu.
- Pour convertir une texture en niveau de gris (comme sur la figure 13-9), vous pouvez donc écrire le pixel shader suivant (la fonction dot fait le produit scalaire de deux vecteurs).

```
float4 pixelShader(PixelInput input) : COLOR
{
    float4 color = tex2D( TextureSampler, input.TexCoord);
    color.rgb = dot(color.rgb, float3(0.3, 0.59, 0.11));
    return( color );
}
```

Sur la figure suivante, la texture appliquée au cube apparaît en niveaux de gris.



Figure 13-9 *Une texture transformée en niveaux de gris*

En résumé

Vous connaissez à présent tout ce qu'il faut pour intégrer des premiers effets à vos jeux, à savoir :

- ce qu'est un shader ;
- la syntaxe du langage HLSL ;
- comment écrire un vertex shader et un pixel shader et comment les intégrer aux jeux.

A

Visual C# Express 2008

Si vous débutez en C#, ou si vous développez habituellement sous Linux, vous n'avez peut-être encore jamais eu à utiliser un outil de la suite Visual Studio.

Cette annexe présente l'interface de Visual C# Express 2008, son éditeur de texte, ses fonctionnalités pour vous faire gagner du temps lorsque vous développez et son débogueur.

Différencier solution et projet

Un projet regroupe un ou plusieurs fichiers et vise la création d'un assemblage (appelé *assembly* en anglais). Un assemblage peut être un fichier .exe, mais aussi une DLL (pour *Dynamic Link Libraries*, bibliothèque de liens dynamiques en français).

Les DLL (attention, il ne s'agit pas forcément de fichiers portant l'extension .dll) correspondent aux projets de type Bibliothèque de classes (figure A-1), ou plus particulièrement pour XNA, aux projets de type « Windows/Xbox 360/Zune Game Library » (figure A-2). Une DLL ne comporte pas de point d'entrée comme la fonction Main, elle ne contient que du code réutilisable par plusieurs applications.

Dans tout cet ouvrage, nous avons travaillé soit avec XNA ou bien en mode console. Nous avons même utilisé le moteur physique FarseerPhysics, un projet créé par un autre développeur et qui générait une DLL.

Une solution est automatiquement créée lorsque l'on crée un projet. Elle peut bien entendu contenir plusieurs projets, comme lorsque vous écrivez une DLL parallèlement à un projet exécutable et que la bibliothèque devra être utilisée dans ce projet. Dans ce cas, il est conseillé de choisir l'option Créer le répertoire pour la solution, ce qui isole les différents projets de la solution dans des sous-répertoires, rendant le tout beaucoup plus lisible.

Nouveau projet		? ×
Types de projets :	Modèles :	00 00 00 00 00
☐ Visual C# └── XNA Game Studio 3.0	Modèles Visual Studio installés	
	Application Projet vide console Mes modèles	•
Projet de création d'une application avec	une interface utilisateur Windows Forms (.NET Framework 3.5)	
Nom : WindowsFormsApplie	cation1	
	ОК	Annuler

Figure A-1

Création d'une DLL .Net

Nouveau projet			<u>?</u> ×
Types <u>d</u> e projets :		Modèle <u>s</u> :	00 0-0- 00 0-0-
E Visual C#	chudia a a	Modèles Visual Studio installés	-
···· XNA Game	: Studio 3.U	Windows Game Windows Game Library (3.0) (3.0) Library (3.0)	
		Zune Game (3.0) Zune Game Library (3.0)	
		Mes modèles	
			-
A project for creat	ing an XNA Frameworl	3.0 Windows game library (.NET Framework 3.5)	
<u>N</u> om :	WindowsGameLibra	у2	
Emplacement :	C:\Documents and S	iettings\Léo\Mes documents\Visual Studio 2008\Projects	Par <u>c</u> ourir
No <u>m</u> de solution :	WindowsGameLibra	y2 🔽 🔽 Créer le réper <u>t</u> oire pour la solution	
		ОК	Annuler

Figure A-2

Création d'une DLL XNA

Personnaliser l'interface

L'interface de Visual C# Express 2008 est composée de plusieurs fenêtres (l'explorateur de solutions, l'explorateur de propriétés, etc.). Toutes ces fenêtres peuvent être remaniées comme vous le désirez : vous pouvez les déplacer, les cacher ou encore en afficher d'autres.

Par exemple, amusez-vous à fermer toutes les fenêtres pour avoir un écran totalement vierge. Allez ensuite dans le menu Affichage et sélection Explorateur de solution ou utilisez le raccourci clavier Ctrl + W, S. Cliquez avec le bouton droit sur la barre de titre de la fenêtre qui s'est ouverte (figure A-3) : vous pouvez définir si elle doit être flottante ou rattachable à un bord de la fenêtre (on parle également de fenêtre « ancrable »).



Pour ancrer une fenêtre à un bord de la fenêtre, maintenez le clic sur sa barre de titre vers la marque du bord désiré, elle s'y accroche automatiquement (figure A-4).

Figure A-4

La fonction d'ancrage des fenêtres est très pratique



Vous avez la possibilité de n'afficher les fenêtres que lorsque votre souris passe sur le bord de la fenêtre. Lorsque vous cliquez avec le bouton droit sur la barre de titre de la fenêtre, choisissez l'option Masquer automatiquement.

Si vous disposez d'un écran assez grand et si vous aimez cette manière de travailler, vous pouvez afficher deux fichiers (ou plus) en parallèle, aussi bien verticalement qu'horizon-talement (figure A-5).



Figure A-5

Avoir plusieurs fichiers ouverts en même temps peut-il améliorer la productivité ?

L'éditeur de texte

L'éditeur de texte de Visual C# Express 2008 est un outil particulièrement paramétrable. La première chose que fait généralement un développeur après avoir installé l'IDE est de choisir d'afficher le numéro des lignes (option qui n'est malheureusement pas activée par défaut).

- Cliquez sur le menu Outils puis sur Options : vous accédez à la fenêtre des options de l'éditeur de texte.
- 2. Dans l'arbre à gauche de la fenêtre, choisissez Éditeur de texte.
- 3. Sélectionnez C#.
- 4. Enfin, cochez la case Numéros de ligne (figure A-6).

Dans cette fenêtre, vous pouvez également définir la taille et le comportement des tabulations, l'affichage des erreurs ou encore le comportement d'IntelliSense.

IntelliSense

IntelliSense est le composant de Microsoft qui assure la saisie automatique de code et qui, lorsque vous écrivez un programme, vous propose une liste contextuelle d'éléments disponibles (propriétés, méthodes, etc.).

Visual C# Express 2008 ANNEXE A 325



Figure A-6 Les options de l'éditeur de texte

Les directives #region et #endregion ne sont pas spécifiques à la suite de Microsoft, mais font partie intégrante du langage C#. Elles masquent une partie du code pour améliorer la lisibilité.

```
#region Test
public class Class1
{
}
#endregion
```

Si vous copiez l'exemple ci-dessous dans l'éditeur de texte de Visual C# Express 2008, vous verrez apparaîtrez un petit signe – à côté de l'expression #region. Lorsque vous cliquez sur ce signe, le contenu est automatiquement masqué et remplacé par le mot Test.

Dans l'éditeur de texte, vous pouvez également choisir de masquer le contenu d'une fonction, d'une classe, d'un espace de noms, etc., sans utiliser les régions (figure A-7).

```
namespace WindowsGameLibrary1
{
    Test
    public class Class2...
}
```

Figure A-7 *L'implémentation de la fonction est cachée*

Pour terminer avec l'éditeur de texte, lorsque vous ajoutez des instructions dans un fichier, une barre verticale jaune apparaît à côté de ces nouvelles lignes (ou des lignes modifiées). La barre deviendra verte une fois que vous aurez sauvegardé votre fichier. Les barres disparaissent dès que le fichier est fermé (figure A-8).

```
16 public class Class1
17 {
18 // Lignes sauvegardées
19 - // Ligne non sauvegardée
20 - }
```

Figure A-8

Changements de couleur de la barre verticale

Les extraits de code

Les extraits de code (appelé *code snippet* en anglais) servent à insérer rapidement du code préconfiguré. L'extrait de code peut être une structure de base du langage (une boucle for par exemple) ou un extrait de code plus complet que vous aurez défini.

Pour insérer un snippet :

- 1. Cliquez avec le bouton droit.
- 2. Sélectionnez Insérez un extrait..., ou bien utilisez le raccourci clavier Ctrl + K, X.
- 3. Choisissez ensuite l'extrait de code qui vous intéresse.

Sur la figure A-9, vous voyez l'ajout d'un extrait de code pour les propriétés. Vous pouvez vous déplacer entre les différentes zones en couleur grâce à la touche Tab.

```
int test;
public int MyProperty { get; set; }
```

Figure A-9 Ajout d'un code snippet

Avec XNA, vous devez souvent réécrire toute un bloc de lignes concernant les directives using... Pour gagner du temps, vous allez donc créer un *code snippet* qui évitera d'avoir à réécrire toutes ces lignes.

L'extrait de code est représenté par un fichier XML qui doit porter l'extension .snippet. Le schéma du fichier est disponible sur le site de MSDN à cette adresse : *http://msdn.microsoft.com/en-us/library/ms171418.aspx*.

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
<CodeSnippet Format="1.0.0">
```

```
<Header>
      <SnippetTypes>
        <SnippetType>Expansion</SnippetType>
      </SnippetTypes>
      <Title>Directives using pour XNA</Title>
      <Shortcut>usingxna</Shortcut>
      <Description>Insere automatiguement les directives using pour XNA.
      ►</Description>
      <Author>leonard |ABAT</Author>
    </Header>
    <Snippet>
      <Code Language="csharp"><![CDATA[using Microsoft.Xna.Framework;</pre>
     using Microsoft.Xna.Framework.Audio:
     using Microsoft.Xna.Framework.Content:
     using Microsoft.Xna.Framework.GamerServices:
     using Microsoft.Xna.Framework.Graphics:
     using Microsoft.Xna.Framework.Input;
     using Microsoft.Xna.Framework.Media:
     using Microsoft.Xna.Framework.Net;
     using Microsoft.Xna.Framework.Storage:]]></Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

< ![CDATA[]]>

Le CDATA évite l'exécution d'une partie du fichier XML. Ce qui se situe dans cette balise ne sera donc pas parsé.

Cliquez ensuite sur Outils, puis sur Gestionnaire des extraits de code, et enfin sur le bouton Importer... Allez chercher le fichier .snippet. L'extrait de code est maintenant prêt à être utilisé.

Refactoriser

Imaginez la situation suivante : vous vous êtes lancé dans la création d'une application (un jeu, une application Windows, etc.) et vous êtes déjà bien avancé. Seulement, au bout d'un moment, vous vous rendez compte que vous avez fait une erreur de design, ou vous n'êtes pas satisfait d'un résultat : les modifications à apporter au projet risquent donc de vous prendre beaucoup de temps.

La refactorisation (de l'anglais *refactoring*) permet de remanier facilement le code. Les différentes possibilités de refactorisation sont accessibles depuis le menu Refactoriser lorsque vous avez sélectionné un élément du code.

Si vous avez lu ce livre en entier, vous avez déjà rencontré une technique de refactorisation : le renommage. En effet, par défaut dans tout projet XNA, la classe de base s'appelle Game1. Si vous avez également créé une variable qui s'appelle Game1Int, et que finalement vous décidez de renommer la classe en MyGame, Visual C# Express 2008 procédera de manière intelligente et ne renommera que la classe et ses utilisations.

Il existe plusieurs autres techniques :

- extraire la méthode, ce qui vous permettra de générer automatiquement une méthode à partir d'instructions que vous aurez sélectionnées;
- extraire l'interface, ce qui vous permettra de créer une interface à partir du nom d'une fonction publique;
- supprimer les paramètres, ce qui vous permettra de supprimer un argument de la signature d'une fonction, etc.

Déboguer une application

Avant de déboguer un projet, vous devez vous assurer que vous le générez en mode *Debug*. Pour cela, vérifiez que c'est bien Debug qui est sélectionné dans la liste Configuration de solutions (figure A-10).

Debug 🗸 x86	
Debug	ij
Release	L
Gestionnaire de configurations	I

Figure A-10

Sélection de la configuration à appliquer

Attention

Avant de distribuer votre application, recompilez-la en mode *Release*. L'assemblage ainsi généré est optimisé et ne comprend pas de données relatives au débogage.

Pour lancer une application et utiliser le débogueur, cliquez sur le menu Déboguer, puis sur Démarrer le débogueur, ou utilisez le raccourci clavier F5. Pour stopper le débogage, retournez dans le même menu et choisissez cette fois Arrêter le débogage, ou utilisez le raccourci clavier Maj + F5.

Les points d'arrêt (*break points* en anglais) sont un élément essentiel du débogueur : vous en placez sur chaque instruction où le programme doit suspendre son exécution. Ils sont donc pratiques pour vérifier que le programme passe bien par certains endroits.

Pour placer un point d'arrêt sur une ligne, cliquez sur la colonne située à gauche de celle-ci ou utilisez le raccourci F9. Le retrait d'un point d'arrêt se fait de la même manière. Lorsque vous avez ajouté un point d'arrêt sur une ligne, celle-ci s'affiche en rouge dans l'éditeur de texte. Placez un point d'arrêt quelque part dans le code (par exemple, dans la méthode Update() si vous travaillez sur un jeu avec XNA) et lancez le débogueur. Dès que le programme atteint le point d'arrêt, la fenêtre de Visual C# Express 2008 passe au premier plan. La ligne où s'est arrêté le programme est maintenant visible en jaune dans l'éditeur de texte (figure A-11).



Figure A-11 *L'exécution s'est suspendue sur le point d'arrêt*

Vous pouvez afficher le contenu d'une variable simplement en passant dessus le curseur de la souris. Pour les objets plus complexes, faites dérouler la liste des informations grâce au signe + (figure A-12).



Figure A-12

Visualisation de la valeur d'une variable

Vous pouvez aussi voir le contenu des variables via les fenêtres espions. Pour ajouter un espion sur une variable, faites un clic droit sur la variable et choisissez Ajouter un espion. Son contenu est alors visible dans la fenêtre Espion (figure A-13). La fenêtre Variables locales présente automatiquement les variables définies dans le contexte courant.

Espion		- ₽ ×
Nom	Valeur	Туре 🔺
📮 🦸 gameTime	{Microsoft.Xna.Framework.GameTime}	Microsoft
	{00:00:00.0166667}	System.T
- 🕀 🚰 ElapsedRealTime	{00:00:00.0205232}	System.T
- IsRunningSlowly	true	bool
🕀 🕀 TotalGameTime	{00:00:03.8166743}	System.T
- 🕀 🚰 TotalRealTime	{00:02:47.2546128}	System.T
🕒 🕀 🧳 Membres non publics		

Figure A-13 Utilisation d'un espion

Il ne s'agit ici que d'une présentation rapide du débogueur de Visual C# Express 2008. Il possède bien d'autres fonctionnalités et le framework .Net dispose aussi des classes Debug et Trace qui pourraient vous être utiles...

Raccourcis clavier utiles

La meilleure façon d'augmenter votre productivité est de maîtriser les différents raccourcis clavier de Visual Studio. Le tableau A-1 énumère les plus couramment utilisés.

Toujours plus de raccourcis

Si vous avez envie d'abandonner complètement votre souris et de devenir un as du clavier, Microsoft a mis à la disposition des développeurs C# un poster qui regroupe tous les raccourcis clavier disponibles dans Visual Studio 2008.

http://www.microsoft.com/downloads/details.aspx?familyid=E5F902A8-5BB5-4CC6-907E-472809749973&displaylang=en

Raccourci	Description
Ctrl + M,O	Masquer une zone.
Ctrl + M,M	Déplier une zone masquée.
Ctrl + E,C	Commenter la zone sélectionnée.
Ctrl + E,U	Décommenter la zone sélectionnée.
ТАВ	Insérer un snippet.
Shift + Alt + C	Afficher la fenêtre d'ajout de classe.
Ctrl + J	Afficher IntelliSense.
Ctrl + F4	Fermer le fichier ouvert.
Ctrl + Shift + B	Générer le projet.
F5	Lancer le débogage.

Tableau A-1 Raccourcis clavier les plus couramment utilisés

B

Les bienfaits de la documentation

Dans cette annexe, vous allez apprendre où trouver de la documentation et comment générer la documentation de vos projets.

L'incontournable MSDN

La première chose à connaître lorsque vous débutez avec XNA ou, d'une manière plus générale, avec la programmation .Net, c'est l'existence de MSDN (*Microsoft Developer Network*). MSDN, c'est tout simplement :

- Un site web, qui est disponible en français (*http://msdn.microsoft.com/fr-fr/*) et sur lequel vous retrouvez des actualités, des dossiers ou encore des événements à propos des technologies Microsoft (figure B-1).
- Une gigantesque bibliothèque de documentations qui contient des tutoriaux et un guide de référence sur toutes les classes du framework .Net ou XNA (figure B-2). Cette base de connaissances est également disponible en téléchargement lorsque vous installez un outil de la suite Visual Studio.
- Des forums de discussion où vous pourrez trouver de l'aide auprès des experts de la communauté.
- Des blogs où les auteurs postent régulièrement leurs dernières trouvailles ou études, et ce, pas uniquement sur des technologies Microsoft.



Figure B-1

Le site web français de MSDN



Figure B-2

La documentation complète du framework est disponible

MSDN propose un abonnement qui permet, entre autres, à ceux qui en bénéficient d'accéder aux tout derniers logiciels de Microsoft en avant-première.

Enfin, MSDNAA (MSDN *Academic Alliance*) est un programme qui permet aux universités et écoles adhérentes de proposer à leurs étudiants l'accès à divers logiciels Microsoft. Si vous êtes étudiant et que votre structure enseignante vous le permet, vous avez par exemple accès à un abonnement étudiant à l'XNA Creator Club pour tester vos jeux sur Xbox 360 (vous ne pourrez cependant pas vendre vos jeux).

Ressources sur le Web

La communauté XNA est grande, les blogs et les sites des gourous du framework sont très nombreux et leur contenu est à la hauteur. Dans le tableau B-1, nous avons essayé de dresser une courte liste de ces ressources, à visiter et explorer régulièrement en profondeur.

Site	Langue	Description
http://ziggyware.com/	En anglais	Actuellement, presque deux cents articles publiés. Une véritable mine d'or.
http://blogs.msdn.com/shawnhar/	En anglais	Le blog de Shawn Hargreaves.
http://msmvps.com/blogs/valentin/default.aspx	En français	Le blog de Valentin Billotte, MVP.
http://www.c2i.fr/	En français	Le site web de Richard Clark, MVP.
http://blog.emmanueldeloget.com/	En français	Le blog d'Emmanuel Deloget.
http://www.xnadevelopment.com/	En anglais	Le site web de George Clingerman, MVP.
http://leonard-labat.blogspot.com/	En français	Le blog de l'auteur de ce livre.

Tableau B-1 Sites et blogs incontournables

MVP

Le terme MVP (*Microsoft Most Valuable Professional*) est un titre décerné chaque année par Microsoft à des professionnels des technologies Microsoft indépendants. Les experts qui reçoivent ce titre sont récompensés pour le partage de leurs connaissances auprès d'autres membres des communautés d'utilisateurs.

Figure B-3

Le logo généralement présent sur le blog ou le site d'un MVP



Vous vous en doutez sûrement, apprendre à programmer sans support de documentation est mission impossible. De la même manière, il est difficilement imaginable que vous puissiez apprendre à vous servir d'une bibliothèque sans pouvoir vous appuyer sur une source de documentation. Vous aurez donc certainement envie, vous aussi, d'utiliser les fonctionnalités de génération de documentation de la suite Visual Studio afin de distribuer vos projets accompagnés de cette source d'informations.

Nous l'avons vu au début du livre, il existe un format de commentaire particulier dédié à la documentation, les trois slashs à la suite :

| ///

À la suite de ces slashs, vous écrivez une balise XML. Le tableau B-2 répertorie une partie des balises existantes.

Balise	Description
<summary></summary>	Utilisée pour donner la description complète d'une classe, fonction, propriété ou variable.
<param/>	Utilisée pour la documentation d'un argument. Vous devez préciser le nom de l'argument. <param name="arg1"/> Argument 1
<returns></returns>	Utilisée pour documenter la valeur de retour d'une fonction.
<value></value>	Utilisée pour documenter une propriété.
<paramref></paramref>	Utilisée pour signaler qu'un mot, utilisé par exemple dans une balise <summary>, est un paramètre de la fonction.</summary>
	<summary>Je parle de <paramref name="arg1"></paramref> <summary></summary></summary>
	<param name="arg1"/> Argument 1
<c></c>	Utilisée pour documenter du code sur une ligne (la différence a vec du texte classique se situe au niveau de l'affichage).
<code></code>	Même chose que <c>, mais supporte plusieurs lignes.</c>
<remarks></remarks>	Utilisée pour ajouter des remarques ou commentaires particuliers.
<exception></exception>	Utilisée pour préciser le type d'exception que la fonction est susceptible de lever.
<exemple></exemple>	Utilisée pour donner un exemple d'utilisation d'un élément.
<see></see>	Utilisée pour créer un lien vers un autre élément.
	<pre><summary>Avez-vous vu <see cref="method">ma méthode</see> ?</summary></pre>
<seealso></seealso>	Utilisée pour créer un lien en bas de page de la documentation.

Tableau B-2 Les balises de documentation

Une fois votre classe correctement documentée, signalez à l'environnement de développement qu'il doit générer un fichier .xml contenant la documentation du projet.

- 1. Cliquez sur le menu Projet, puis sur Propriétés de ...
- 2. Allez ensuite sur l'onglet Générer.
- 3. Cochez la case Fichier de documentation XML (figure B-4).

Sortie		
Chemin de sortie :	bin\x86\Debug\	Parcourir
Fichier de documentation XML :	bin\x86\Debug\ChapitreDix_2.XML	
🔲 Inscrire pour COM Interop		
Générer un assembly de sérialisation :	Auto	
		Options avancées

Figure B-4

Activation de la génération de documentation

4. Générez ensuite le projet et rendez-vous dans son répertoire de sortie : un fichier .xml a bien été créé.

```
<?xml version="1.0" ?>
- <doc>
- <assembly>
< name>ChapitreDix_2</name>
</assembly>
- <members>
- <member name="M:ChapitreDix_2.Program.Main(System.String[])">
<summary>The main entry point for the application.</summary>
</member>
</members>
</doc>
```

Pour transformer votre documentation en une version plus lisible, essayez par exemple l'utilitaire SandCastle (*http://www.codeplex.com/Sandcastle*) qui génère des fichiers HTML dans le style MSDN (figure B-5).

ChapitreDix	2 Members - Mozilla Firefox		LOA
ichier Edition	Affichage Historique Marque-pages Qutils ?		
<>>- (C 🗙 👝 🗋 File:///C:/Program Files/Sandcastle/Exan	nples/Test/vs2005/chm/html/AllMembers_T_ChaptreDix_2_ChaptreDix_2.htr 🏠 🔹 🚺 🕯 Google	P
Collapse All .NET Frames Chapitre ChapitreDia	B Code: All B rork Class Library Dix_2 Members - 2 Class Constructors Methods Fields Properties Even	nts See Also Send Feedback	
The ChapitreD	bix 2 type exposes the following members.		2
= Constrau	tors		
= consulut			
	Name	Description	_
1 9	ChapitreDix_2		
4 Methods	Name BeginDraw	Description (Inherited from Game.)	
	BeginBup	(Interited from Fiame.)	
	Dispose	Overloaded.	
49	Draw	(Overrides GameDraw(GameTime).)	
46	EndDraw	(Inherited from Game.)	
40	EndRun	(Inherited from Game.)	
19	Equals	(Inherited from Object.)	
-9	Exit	(Inherited from Game.)	
49	Finalize	(Inherited from Game.)	
ο φ	GetHashCode	(Inherited from Object.)	
-4	GetType	(Inherited from Object.)	
4 9	Initialize	(Overrides GameInitialize().)	
40	LoadContent	(Overrides GameLoadContent().)	
4. Ve	LoadGraphicsContent	Obsolete. (Inherited from Game.)	
49	MemberwiseClone	(Inherited from Object.)	

Figure B-5

Documentation générée avec SandCastle

Index

Numériques

3ds max 299

Α

A* 186, 187 abonnement 228 accesseur 179 accolade 16 ActiveSongChanged 144 addition 9 affichage 287 fil de fer 313 algorithme A* 187 recherche de chemin 186 test 197, 204 alias, espace de noms 19 alpha 216 ambient light 294 AmbientLightColor 296 animation 108 sprite sheet 108 ApplyForce 224 arrière-plan 86, 105 aspectRatio 266 assemblage 321 assembly 321 Asteroids 207 asynchrone 144 méthode 154 traitement 155 attribut Serializable 178 speed 210 XmlIgnore 179 AudioEngine 136 autocomplétion 32

backface culling 271, 272 Background 87 balises 146 documentation 334 banque de sons 133 BasicEffect 263, 264, 267, 298, 304 BeginShowKeyboardInput 156 BeginShowMessage 154 bibliothèque 29 multimédia 142 de classes 321 BinaryFormatter 180 Blender 299 bloc catch 150 finally 150 blur 316 Body 224 BodyFactory 224 boîte de dialogue 154 booléen 6, 155 IsTrialMode 160 SimulateTrialMode 160 boucle 55 do 55 for 56, 196 foreach 58, 196 infinie 56 while 55, 196 brouillard 298 buffer 139, 172 vider 175 Buttons 72 ButtonState 71 byte 191

В

С

callback 154 caméra 233, 265, 278 champ de vision 233 déplacer 277 multijoueur 238 Rectangle 233 carte bords 235 graphique 303 case 190, 192 coût 192 cast 180 casting 77 casual (jeu) 35 catch 150 CDATA 327 chaîne de caractères 8 concaténer 24 champ de vision 265 char 7 classe 18, 20 Background 87 BasicEffect 263, 264, 298, 304 Body 224 BodyFactory 224 de test 110 Directory 169 DrawableGameComponent 86 droits d'accès 21 Exception 150 File 172. 174 Game 75 Geom 227 Guide 152, 158 hiérarchie 54 KeyboardState 68 Map 191

classe (suite) MathHelper 266 Matrix 291 MediaLibrary 142 MediaPlayer 142 mère 51 NetworkSession 251 Node 192 NodeList 195 partielle 22 Path 171 PathFinding 196 PhysicsSimulator 222 Player 228 ServiceHelper 88 SignedInGamer 158 sprite 37, 45 SpriteBatch 99, 304 statique 19 Tile 190, 201 WaveBank 138 clavier 67, 75 déplacement du sprite 69 récupérer l'état 68 Clear 214, 244 Clingerman, George 105, 108 Close 175 code snippet 326 CodePlex 219 collection 58 Components 87 générique 58 List 194, 211 ServiceHelper 201 collision 200, 213 détection 91 pixel 215 Player 228 rectangle 213 CollisionPerPixels 216 Color 31, 112, 279 Combine 171 commentaire 11 Components 87 compression 144 condition 11.56 configuration utilisateur 146 const 8 constante 8 constructeur 19, 22, 51 argument 23 Vector2 39

ContainsKey 62 conteneur de Références 165 Content Manager 37, 55, 136 HLSL 305 pipeline 29 Processor 144 ContentImporter 181 continue 56 coordonnées 259, 281 Copy 172 couleur 279 moduler 40 négatif 315 netteté 315 texture 318 Vector4 262 Count 196 Create 172, 249 CreateLookAt 266 CreatePerspectiveFieldOfView 265 CreateRectangleBody 224 CreateRectangleGeom 227 CreateRotationY 291 CreateScale 291 CreateTranslation 293 culling 313 CullMode 271 CurrentTechnique 312

D

déboguer 328 délégué 228 Delete 172 démo 160 désérialisation 148 Deserialize 180 détection, collision 91 dictionary 59 dictionnaire 62 diffuse lighting 294 Dijkstra 186 directive, using 68, 326 Directory 19, 169 DirectX 29 Dispose 253 distance de Manhattan 188, 192 division 9 par zéro 149 **DLL 321** do, boucle 55

documentation 334 balise 334 données extraire 253 récevoir 253 récupérer 253 dossier, jeu 168 dot 318 Draw 30, 40, 48, 99 surcharge 40 DrawableGameComponent 86, 128 DrawUserIndexedPrimitives 272 droits d'accès 21

Е

échelle, modifier 291, 293 écran, partage 231 écrire dans un fichier 174, 175 éditeur de cartes 162 de texte 324 effet 263 fichier 310 flou 316 négatif 315 netteté 315 niveau de gris 318 prise en charge de la lumière 296 technique 264 vague 314 else 12 else if 13 encapsulation 21 énumération 68 Buttons 72 ButtonState 71 CullMode 271 Keys 68 PlayerIndex 71, 238 PrimitiveType 261 SpriteEffects 116 TileType 192 espace de noms 19, 169, 178, 179 déclaration 68 Microsoft.Xna.Framework.Input 68 de stockage 145 dossier de l'utilisateur 146 dossier du jeu 145 espion 329

338

état, souris 201 événement 228 ActiveSongChanged 144 exception 148, 150 personnaliser 151 exe 321 expédition, méthode 253 extrait de code 326

F

factoriser le code 14 far plane 264 FarseerPhysics 207, 219 fenêtre 208 feuille de sprites 109 fichier écrire 174, 175 lire 177 manipuler 172 sérialiser 178 field of view 265 fil de fer 313 File 19, 172, 174 finally 150 Find 251 float 116 floatLxC 307 flou 316 Flush 175 flux 174, 175, 176 FogEnabled 298 fonction 15.17 BeginShowKeyboardInput 156 CollisionPerPixels 216 définir 17 dot 318 mathématique 224 override 51 ResetPosition 229 SetVibration 72 for. boucle 56, 196 force 224 foreach, boucle 58, 196 format CSV 147 INI 148 mp3 133 way 133 XML 146 FPS (Frames Per Second) 128 frame 46

framework 29 FX Composer 305

G

gâchette analogique 72 Game 75 GamePadState 71 gameplay 80 Gamer 158 gamer services 249 tag 249 Gamer Services 152 GamerCard, afficher 158 GamerServicesNotAvailableExce ption 148 GameThumbnail 61 GameTime 88 Geom 227 gestionnaire d'exceptions 148 de contenu 125, 144 image 55, 60 get 22 GetData 215 GetLength 57 GetPressedKeys 69 **GLSL 304 GPU 304** graphics 32, 41, 208 GraphicsDevice 244, 267 gravité 222 GUI 80.114 wWinForms 81 XNA Simple Gui 81 Guide 152, 158

Н

Half Life 2 217 Hargreaves, Shawn 128 Havok 217 héritage 51, 52 multiple 50 HLSL (High Level Shader Language) 304

I

IAsyncResult 155 if 11 image, gestionnaire 55, 60 Imagine Cup 29

IMouseService 200 incrémentation 10 index 271 index buffer 271, 272 inertie 227 Initialize 30, 41, 42, 210, 211 InitializeVertices 279, 295 insertion dichotomique 194 instancier, constructeur 22 instruction continue 56 intelligence artificielle 185 IntelliSense 32, 324 interface 74 extraire 328 **IMouseService 200** règle de nommage 74 IsConnected 71 IsDataAvailable 253 IsTrialMode 160

J

jeu de rôle 50 en réseau 248 Mario 218 plates-formes 218 tile-based 37 Join 251 joueur connecté 158

Κ

Keyboard 68 KeyboardState 68 KeyHasBeenPressed 232 Keys 68 Keys.Escape 69

L

layerDepth 116 lecteur réseau 146 licence 219 lire un fichier 177 List 211 liste 194 fermée 187 générique 195 ouverte 187, 194 triée 194 vider 214 Live, compte 248 Load 299 LoadContent 30, 48, 61, 211, 272 lumière 294 d'ambiance 294, 296, 312 diffuse 294, 296 directionnelle 296 spéculaire 294, 297

Μ

Main 15 manette 67, 71, 238 gâchette analogique 72 multijoueur 71 pad directionnel 72 stick analogique 72 vibration 72 Manhattan 192 Map 191 Mario 218 Math 19, 20 MathHelper 266 matrice 263 contenu 307 de rotation 291 de vue 266 définir 306 résultat 266 Matrix 263, 291, 307 MediaLibrary 142 MediaPlayer 142, 144 mémoire 3 mesh 299 méthode ApplyForce 224 asynchrone 154, 155, 228 BeginShowMessage 154 Clear 214, 244 Close 175 Combine 171 Copy 172 Create 172, 249 CreateLookAt 266 CreatePerspectiveFieldOfView 265 CreateRectangleBody 224 CreateRectangleGeom 227 CreateRotationY 291 CreateTranslation 293 Delete 172 Deserialize 180 Dispose 253 Draw 30, 40, 48, 99

272 extraire 328 Flush 175 générique 77 GetData 215 GetLength 57 GetPressedKeys 69 Initialize 30, 211 InitializeVertices 295 Join 251 KeyHasBeenPressed 232 Load 299 LoadContent 30, 48, 61, 211 Next 210 Open 174 Play 144 PlayCue 136 privée, Initialize 210 ReadType 253 ReceiveData 253 ResetPosition 210 SendData 253 ShowGamerCard 158 statique 169 Find 251 synchrone 251 UnloadContent 30 Update 30, 42, 48, 71, 208, 209, 211, 214, 278, 291 ValidCoordinates 191 Write 175 WriteLine 175 Microsoft XNA Framework Gam e.dll 29 mise à l'échelle 263 mode démonstration 160 Release 149 modèle 299 ajouter au projet 299 charger 299 modeleur 3D 299 molette 70, 71 mot-clé override 51 throw 151 moteur graphique 29 physique 207, 217 MouseState 70

DrawUserIndexedPrimitives

MSDN (Microsoft Developer Network) 331 MSDNAA (MSDN Academic Alliance) 333 msElapsed 203 multijoueur 237 multiplication 9

Ν

near plane 264 netteté 315 NetworkSession 249, 251, 253 NetworkSessionProperties 251 Next 210 nextColor 112 niveau 187 de gris 318 Node 192 NodeList 195 nœud 187, 192 coût de passage 189, 200 nombre aléatoire 210 entier 4 réel 6 notation pointée 307

0

objet 18 AudioEngine 136 BinaryFormatter 180 ContentManager 62 Cue 137 GamePadState 71 Gamer 158 GameTime 88 graphics 41, 208 Keyboard 68 MouseState 70 random 210 SoundBank 136 SpriteBatch 128 SpriteFont 128 Spritefont 125 StorageContainer 153, 168 StreamWriter 175 String 19 TimeSpan 211 Vector2 222 WaveBank 136 Open 174

OpenGL 304 opérateur && 216 conditionnel 12 incrémentation 10 logique 14 ternaire 214 override 51

Ρ

PacketReader 253 PacketWriter 253 pad directionnel 72 paquet 249 paramètre callback 154 PlayerIndex 158 supprimer 328 partial 22 passe 264, 310 Path 171 Pathfinding 186, 196 performances 63, 253, 266 temps d'exécution 63 utilisation mémoire 63 périphérique 67.154 clavier 67 manette 67, 71 souris 70, 200, 201 spécialisé 73 batterie 73 guitare 73 tapis de danse 73 perspective, projection 264 Phun 218 PhysicsSimulator 222 physique 217 piste. Cue 136 pixel shader 304, 308, 310 plan éloigné 264 proche 264 PlayCue 136 Plaver 228 PlayerIndex 71, 158, 238 plein écran 33 point d'arrêt 328 police de caractères 126 Pong 83, 231 Portal 217 Position 281

preset 140 primitive 261 type 261 PrimitiveType 261 private 22 procédure 15 déclarer 16 nom 16 processus asynchrone 251 projection en perspective 264 orthogonale 265 projet 321 partager 29 planifier 83 Pong 83 pseudo-code 84 sonore. créer 132 propriété AmbientLightColor 296 Count 196 FogEnabled 298 Game 87 IsConnected 71 IsDataAvailable 253 ScrollWheelValue 71 Services 75 SpecularColor 297 Techniques 312 TitleLocation 168 pseudo-code 84 public 22

R

raccourcis clavier 330 random 210 rastérisation 304, 308 ReadType 253 ReceiveData 253 rectangle 214 caméra 233 refactoriser 327 référence 93 Release 149 rendu, shader 303 renommage 327 repère, main 260 réseau 248 quitter une partie 253 ResetPosition 210, 229 réverbération 140

Role Playing Game 50 rotation 116, 263 matrice 291 point d'origine 118

S

SandCastle 335 sauvegarde, StorageContainer 153 SavedGames 146 scrolling 105, 232 classe de test 106 ScrollWheelValue 71 sémantique 308 SendData 253 séparation du code 30 sérialisation 148 sérialiser 178 en binaire 178 un fichier 178 XML 179 Serializable 178 service état de la souris 201 XNA 73, 75 ServiceHelper 88, 201 session 249 set 22 SetVibration 72 shader 303 ShowGamerCard 158 SignedInGamer 158 SimulateTrialMode 160 Socket 19 solo 232 solution 321 son API SoundEffect 141 **XACT 131** bonnes pratiques 144 compression 139 ininterrompu 139 lire 136, 139, 141 streaming 138 SoundBank 136 SoundEffect 141 lire musique 142 son 141 MediaPlayer (classe) 142 Zune 141 sourceRectangle 213

souris 70 curseur 71 molette 71 soustraction 9 specular light 294 SpecularColor 297 speed 210 splitté 231 sprite 36, 40 afficher 37.47 animation 108 classe 37.45 déplacement au clavier 69 échelle 119 inversion 120 mouvement 41 ordre d'affichage 124 profondeur d'affichage 116, 124 rotation 116 sheet 108 sprite sheet 108 transformation 116 SpriteBatch 86, 99, 128, 304 SpriteEffects 116 SpriteFont 128 Spritefont 125 starter kit 26 statique méthode 265, 266 variable 209 stick analogique 72 Stopwatch 64 StorageContainer 153, 168 STR 190 stratégie en temps réel 190 streaming 138 StreamWriter 175 string 8 structure 31, 308 de contrôle 308 Vector2 39 **VertexPositionNormalTexture** 295 surcharge 39, 99 switch 242 swizzling 307 synchrone 154

System.Diagnostics 64 System.IO 169 System.Runtime.Serialization.For matters.Binary 178 System.XML.Serialization 179

Т

tableau 57 byte 191, 235 Color 215 de tableaux 307 multi-dimensionnel 57 parcourir 57, 58 tileList 233 Tales of Phantasia 37 technique 310 effet 264 passe 264 Techniques 312 test 11, 197, 249 combiner 14 condition 13 test.sav 175 Tetris 35 texte FPS (Frames Per Second) 128 optimisation 129 police de caractères 126 Spritefont 125 texture 46, 281 à cheval 291 appliquer 100 arrière-plan 105 associer à une couleur 284 coordonnées 281 couleur 318 différente selon face 289 flouter 316 n'afficher qu'une partie 283 négatif 315 netteté 315 portion 102 problème 287 redimensionner 100 répéter 283 scrolling 105 teinte, varier 112 TextureCoordinate 281 TextureEnabled 282

TextureManager 62 texturer 99, 285 this 23 throw 151 Tile 190, 201 TileType 192 TimeSpan 19, 211 TitleLocation 168 ToRadians 266 transformation 291 translation 263, 293 trial mode 160 TrueSpace 299 try ... catch 158

U

UnloadContent 30 Update 30, 42, 48, 71, 208, 209, 211, 214, 278, 291 using 19, 326

۷

ValidCoordinates 191 variable 4 aspectRatio 266 sourceRectangle 213 statique 209 vecteur 39, 41, 262 composant 307 Vector2 39, 46, 116, 222, 262 constructeur 39 Vector3 262, 273 vertex 260 couleur 279 ordre d'affichage 271, 272 shader 304. 309 VertexDeclaration 282 VertexOutput 309 VertexPositionColor 267, 279 VertexPositionColorTexture 284 VertexPositionNormalTexture 295 VertexPositionTexture 281 vibration de la manette 72 Viewport 232, 237, 241 Visual Studio 321 Viterbi 186 vitesse, déplacement du personnage 203 void 16, 17

342

Index

vue 244 affichage classique 246 dessiner 238 matrice 266 nettoyer le contenu 244 réutiliser 248

W

WaveBank 136, 138 buffer 139 offset 139 while 196 while (boucle) 55 Wii Sport 35 Windows Game Library 162 Windows Media Player 142 Write 175 WriteLine 175 wWinForms 81

Х

XACT 131 compression 139 ADPCM (Windows) 139 XMA (Xbox 360) 139 Cue 133 formats supportés 133 réverbération 140 Sound Bank 133 streaming 138 Wave 133 Wave Bank 133 XACT Auditioning Utility 135 Xbox LIVE 152 Xbox Live 29 Xbox, BeginShowStorageDeviceSele ctor 153
Xbox 360 transfert de jeu 33
XML (eXtensible Markup Language) 146, 334 désérialiser 181 sérialiser 179
XmlIgnore 179
XMA Simple Gui 81
XNB 38 xnb 181

Ζ

Zelda 37 Zune 27, 132, 146

Développement XNA pour la Xbox et le PC

Grâce au tandem Live et XNA, la programmation de jeu vidéo pour PC et Xbox 360 est accessible au plus grand nombre : il n'est plus nécessaire d'investir dans de ruineux outils pour donner libre cours à ses idées de jeux et les réaliser. Cet ouvrage permettra au lecteur de s'approprier le framework XNA 3.0, mais également de comprendre comment s'organise un projet de développement de jeu vidéo.

Accéder aux dernières technologies de développement PC et Xbox 360 avec le framework XNA 3.0

Pour accompagner l'explosion du développement amateur favorisé par la plate-forme de distribution en ligne Live, Microsoft a mis au point le framework XNA pour fournir toutes les briques nécessaires à la création de jeu vidéo. Supports de référence du Live, Xbox 360 et PC sont, grâce à XNA, les deux plates-formes les plus propices pour les studios indépendants, les freelances et les particuliers qui souhaitent faire connaître, voire commercialiser, leurs réalisations.

Un manuel complet pour se lancer dans un projet de création de jeu vidéo

Ce livre accompagne le lecteur, débutant ou non, dans la conduite d'un projet de jeu en C#, qu'il s'agisse de programmer des événements, de créer un environnement sonore, ou de choisir ses moteurs graphique et physique et de les exploiter. L'auteur y détaille les techniques de programmation 2D et 3D. Il explore également les techniques graphiques et sonores avancées (effets, textures, défilement, transformations, animation, éclairage, design sonore, streaming) mais aussi certains algorithmes d'intelligence artificielle, sans oublier l'inclusion du mode multijoueur en réseau ou en écran splitté.

Au sommaire

XNA et son environnement • Débuter en C# • Types de données • Commenter le code • Conditions • Fonctions et procédures • Classes et espace de noms • Prise en main • EDI • Starter kit • Architecture d'un projet XNA • Créer un projet • Outils pour la Xbox 360 • Les sprites • Afficher plusieurs sprites • La classe Sprite • Gestionnaire d'images : boucles, tableaux et collections • Mesure des performances • Interaction avec le joueur • Périphériques • Services • GUI • Programmer un Pong • Pseudo-code • Création du projet • Arrière-plan, raquette, balle • Améliorer le jeu • Textures, défilement, animation • Texturer un rectangle • Scrolling • Sprites sheets • Variation de teinte • Transformations • Spritefont • Sonorisation • XACT et SoundEffect • Créer un projet sonore • Lire un son et un morceau de musique • Streaming • Design sonore • Exceptions et gestion des fichiers : sauvegarder et charger un niveau • Espace de stockage • Sérialisation • Exceptions • Gamer Services • Un éditeur de cartes • Content Importers • Version démo • Pathfinding : programmer le déplacement des personnages • Algorithme et intelligence artificielle • Implémenter l'algorithme A* • Collisions et physique • Zone de collision • Moteur physique • Mode multijoueur • Partager l'écran • Gestion des caméras • En réseau avec Live • Programmation 3D • Coordonnées, primitives, vertices, vecteurs, matrices, transformations, effets, projection • Caméras • Matrices de vue et de projection • Appliquer une couleur à un vertex • Plaguer une texture • Transformations des objets • Lumières • Éclairer la scène • Exploiter les modèles • Améliorer le rendu avec le High Level Shader Language • Vertex shaders et pixel shaders • Syntaxe du HLSL • Fichier d'effet • Ondulation • Textures : en négatif, netteté, flou, couleur • Annexes • Visual C# Express 2008 • La documentation.

À qui s'adresse cet ouvrage ?

 Aux étudiants en programmation qui désirent adapter leurs connaissances aux spécificités du développement de jeu pour PC et Xbox.

- Aux studios indépendants et freelances qui souhaitent passer à XNA.
- À l'amateur curieux qui a choisi XNA pour développer son premier jeu.



L. Labat

Passionné par le développement et les jeux vidéo, Léonard Labat assure une veille sur les technologies Microsoft en publiant régulièrement sur son blog (http://leonard-labat.blogspot. com/). Il évolue au sein du laboratoire des technologies .Net de SUPINFO (http://www.labo-dotnet. com/).

