Hands-On Lab

Créer votre premier jeu

Version: 1.0.0

Dernière mise à jour : 1/10/2012

Sommaire

Vue d'ensemble

Cet atelier va vous permettre de vous familiariser aussi bien au développement de jeux vidéo à l'aide du XNA Game Studio, pour Windows Phone et plus généralement pour toutes les plateformes ciblées par le Framework XNA.

En effet, à travers le développement d'un jeu très simple vous apprendrez les bases du développement à l'aide du XNA Game Studio ainsi que l'utilisation de Microsoft Visual Studio 2010 Express pour Windows Phone afin de créer des jeux pour Windows Phone 7.

Objectifs

A la fin de cet atelier, vous aurez appris :

- Comment est constituée l'architecture d'un moteur de jeu pour Windows Phone 7 développé avec le XNA Game Studio
- Comment ajouter et utiliser des ressources (images, police de caractères, etc.) au sein de vos jeux XNA Game Studio
- Comment développer la logique d'un jeu XNA Game Studio
- Comment fonctionne l'affichage et le rafraîchissement d'un jeu Windows Phone 7 développé avec le XNA Game Studio.

Pré requis

Afin de mener à bien cet atelier, vous devez installer les éléments suivants :

- Microsoft Visual Studio 2010 Express pour Windows Phone 7

Exercices

Cet atelier est divisé en plusieurs exercices. Ceux-ci sont listés ci-dessous :

1. Base du développement de jeu avec XNA
2. Utilisation de ressource dans un jeu XNA
3. La boucle de jeu XNA
4. Gestion des entrée dans un jeu développé avec XNA Game Studio
5. Etude de cas : la logique du jeu « Alien Game »

Temps estimé pour compléter cet atelier: **60 minutes**.

Atelier : Créer votre premier jeu

Microsoft® XNA™ Game Studio 4.0 est un produit vous permettant de développer très simplement vos propres jeux vidéo. Tout développeur, qu'il soit étudiant, passionné ou professionnel peut utiliser cet outil pour développer des jeux et partager ses création.

Le XNA Game Studio 4.0 est un outil fournit par Microsoft et pensé pour être utilisé avec Visual Studio 2010 Express pour Windows Phone, permettant ainsi aux développeurs d'allier la puissance et la simplicité du langage C# pour développer des jeux vidéo.

XNA Game Studio 4.0 embarque le Framework XNA en tant que tel ainsi que le XNA Framework Content Pipeline qui permet d'importer et d'utiliser très simplement des modèles 3D, texture, sons ou autres médias au sein de vos jeux. Il fournit également une API (Application Programming Interface) orientée pour le développement de jeux vidéo, permettant ainsi de simplifier et unifier les développements pour ses différentes plateformes cibles : Xbox 360®, Windows® et depuis peu **Windows Phone 7 Series®**.

Bien que XNA soit un Framework permettant d'accélérer le développement de jeux vidéo, il ne s'agit pas d'une solution permettant de glisser / déposer des composants et de les lier entre eux pour obtenir un jeu. Il est tout d'abord primordial de bien comprendre les bases du développement de jeux vidéo avant de se lancer dans le code.

Ce qu'il est important de comprendre, c'est que le Framework XNA n'est pas un moteur de jeu dans le sens où il ne fournit pas nativement des outils permettant de représenter des éléments, détecter et gérer des collisions ou encore toute autres notions fondamentales dans le développement d'un jeu. C'est vous, développeurs, qui êtes seuls responsable et maîtres de la façon dont le jeu va fonctionner.

Tout au long des exercices de cet atelier, vous apprendrez à prendre en main le développement avec le XNA Game Studio pour Windows Phone au travers de la création d'un jeu, « Alien Game », un jeu de tir des plus simplistes. Le but de ce célèbre jeu est de protéger la Terre d'une invasion extra-terrestre le plus longtemps possible. Plus vous résistez à l'invasion, plus la difficulté augmente !

Vue d'ensemble de l'architecture

« Alien Game » utilise le projet « Game State Management Sample » pour gérer les différents écrans affichés dans le jeu (vous pouvez retrouver ce projet sur cette page : http://creators.xna.com/en-US/samples/gamestatemanagement). Il existe trois écrans au total, pour chacun des états suivants :

- Menu Principal (classe **MainMenuScreen**)
- Jeu en cours (classe **GameplayScreen**)
- En pause (classe **PauseScreen**)

Tout le contenu du jeu est chargé lors du démarrage de celui-ci. Le premier élément à charger et afficher est le fond d'écran du jeu : « **BackgroundScreen** ». C'est ce dernier qui, une fois chargé va

s'occuper d'instancier, charger et afficher l'écran de chargement du jeu : « **LoadingScreen** ». A son tour, l'écran de chargement va avoir la responsabilité de charger le contenu du jeu.

Le fond d'écran n'est responsable d'aucun affichage particulier, par contre, l'écran de chargement va pouvoir afficher diverse informations relatives à la progression du chargement du jeu. En règle général, on préfèrera ne pas surcharger cet écran et ce dans le but d'augmenter la vitesse de chargement du jeu, ce qui intéresse vraiment l'utilisateur.

Dès lors que le contenu du jeu est chargé, le menu principal est affiché à l'écran et les éléments du menu s'affichent de manière animée. L'utilisation d'animation permet de faire paraître plus rapides et moins abruptes les transitions entre les différents écrans du jeu.

Il est important de charger les composant du jeu au démarrage de celui-ci, d'autant plus sur des petits périphériques tels que des terminaux Windows Phone. Ceux-ci n'ayant pas les mêmes capacités matérielles qu'un ordinateur ou une Xbox, charger les éléments entre les différents écrans pourraient ennuyer le joueur qui préfèrera un jeu qui charge tout d'un coup et qui reste fluide par la suite.

Les classe GameplayScreen et Game

L'architecture du jeu « Alien Game » est assez simple. Toute la logique du jeu ainsi que le dessin des éléments à l'écran est réalisé dans la classe **GameplayScreen.** Dans le fichier « GameplayScreen.cs », trois types spécifiques au jeu sont utilisés : **Bullet**, **Alien** et **Player.** Tout le code écrit dans ce fichier est regroupé par fonctionnalités : chargement et déchargement du contenu, mise à jour et simulation du jeu, et enfin dessin à l'écran.

Classe ParticleSystem

Le jeu « Alien Game » utilise un système de particule très simple, basé sur les « sprites » XNA pour la réalisation des effets d'explosion et de traînée du jeu. La définition et la création de ces effets est codée en dure dans la classe **ParticleSystem** et ceux-ci sont accessible par le biais d'une « factory » offrant des méthodes du type « **CreateXXXEffect** ».

L'image ci-dessous représente le rendu final du jeu :

Figure 1

Jeu « Allien Game » en cours d'exécution sur un Windows Phone.

Bases d'un jeu développé avec XNA Game Studio

Le jeu est développé en différentes couches qui sont interconnectées entres elle : but du jeu, joueur, ennemis, intelligence etc. Pour un cas simple, chaque couche peut être traitée comme si c'était un jeu complet et peut en général être décomposée en 3 états :

- **Chargement** – C'est lors de cet état que le système va charger toutes les ressources associées au jeu, définir les différentes variables utilisées, l'environnement initial du jeu (logique dans laquelle tout le jeu va s'exécuter) ainsi que toutes autres tâches nécessaire à la bonne initialisation du jeu. Le jeu passe par cet état une seule fois au cours de son cycle de vie.
- **Mise à jour** – C'est dans cet état que le système va calculer toutes les modifications à apporter aux variables du jeu afin de modifier son état courant : position du joueur, actions, déplacements des ennemis, etc. Tous ces éléments seront basés sur l'intelligence du jeu en tant que telle ainsi que sur l'intelligence du joueur, matérialisée par les entrées (orientation du terminal, appui sur une touche…). Cette étape est répétée à intervalle régulier pendant toute la durée de la partie.
- **Dessin** – Dans cette étape, le système va se charger de dessiner à l'écran le jeu dans son état actuel, et ce en fonction des modifications apportées aux variables de positions, d'actions (etc.) calculées lors de la phase de mise à jours. Tout comme celle-ci, cet état se répète tout au long du cycle de vie du jeu.

Les deux dernières étapes font partie de la boucle de jeu qui est proposée par le Framework XNA. Elles sont exécutées environ 60 fois par seconde sur un PC ou une Xbox 360 et environ 30 fois sur des terminaux moins puissants tels que le Zune, le Zune HD ou les terminaux Windows Phone 7.

Exercice 1 - Base du développement de jeu avec XNA

Dans cet exercice, vous allez apprendre à développer votre premier jeu pour Windows Phone à l'aide du XNA Game Studio. Le jeu sera des plus simples pour commencer, mais vous l'etofferez tout au long de la réalisation des différents exercices qui composent l'atelier.

1. Démarrer Visual Studio 2010 Express pour Windows Phone

   > **Remarque:** toutes les étapes décrites dans cet atelier s'appuient sur la version Express de Visual Studio 2010 pour Windows Phone, mais il est tout à fait possible d'utiliser une version complète de Visual Studio 2010 accompagnée des outils de développement pour Windows Phone.

2. Pour ouvrir Microsoft Visual Phone Developer 2010 Express rendez-vous dans le menu **Démarrer | Tous les programmes | Microsoft Visual Studio 2010 Express**.

   > **Visual Studio 2010:** Pour ouvrir Visual Studio 2010 rendez-vous dans le menu **Démarrer | Tous les programmes| Microsoft Visual Studio 2010**.

3. Dans le menu **File**, cliquez sur **New Project**.

   > **Visual Studio 2010:** Dans le menu File, pointez **New** et cliquez sur **Project**.

4. Dans la fenêtre **New Project**, choisissez la catégorie **XNA Game Studio 4.0** et selectionnez **Windows Phone Game (4.0)** dans la liste des templates installés. Nommez le projet **AlienGame** et cliquez sur **OK**.

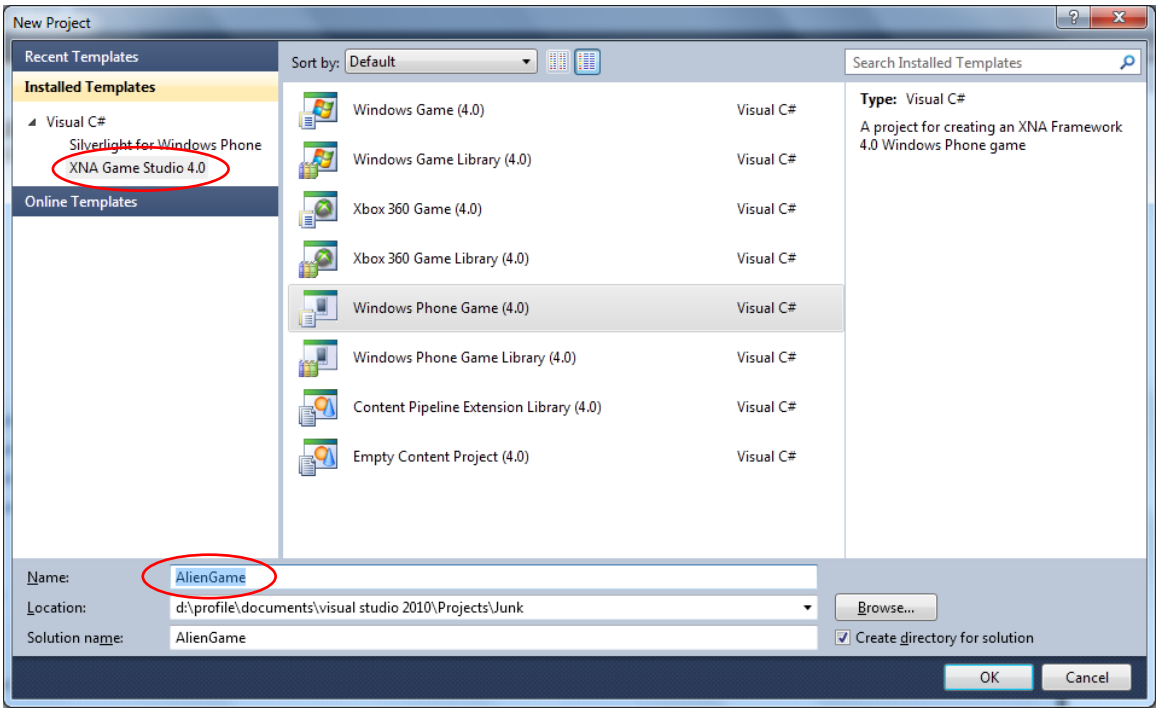


Figure 2

Création d'un nouveau projet de type « Windows Phone Game (4.0) » dans Visual Studio 2010 Express pour Windows Phone 7.

5. Dans le **Solution Explorer**, passez en revu la solution qui a été générée par Visual Studio lors de la creation du projet. Une solution Visual Studio est un conteneur de projets. Dans le cas présents, deux projets ont été générés : **AlienGame** et **AlienGameContent**, le premier de type XNA Game Studio game for Windows Phone et le second permettant de stocker les ressources associées au jeu :

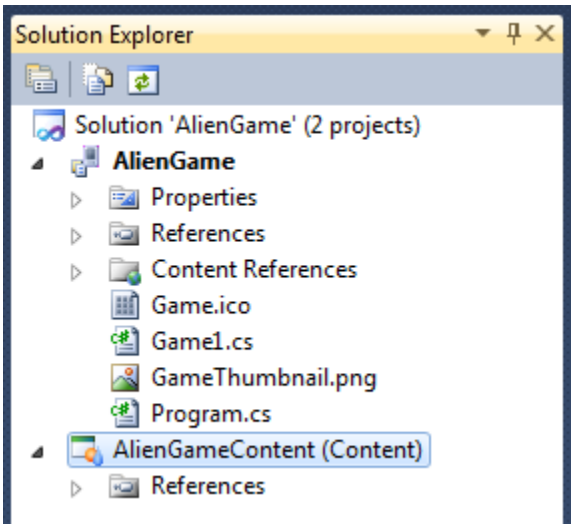


Figure 3

Application AlienGame dans l'explorateur de solution

> **Remarque :** L'explorateur de solution vous permet de visualiser les différents éléments qui composent votre solution ainsi que de gérer ceux-ci. Pour afficher la fenêtre **Solution Explorer** pressez **CTRL + W, S** ou rendez-vous dans le menu **View** et cliquez sur **Other Windows | Solution Explorer**.

6. Dans le projet généré, vous pouvez retrouver une impplémentation basique du jeu, et notamment la boucle de jeu. Pour visualiser cette implémentation, rendez-vous dans le fichier Game1.cs.
7. Ouvrez le fichier Game1.cs. Il est recommandé de renommer le jeu par défaut et de lui donner un nom plus représentatif de la réalité.
8. Renommer la classe générée par défaut (Game1) en AlienGame. Pour se faire, faites un clic droit sur le nom de la classe et sélectionnez Refactor -> Rename

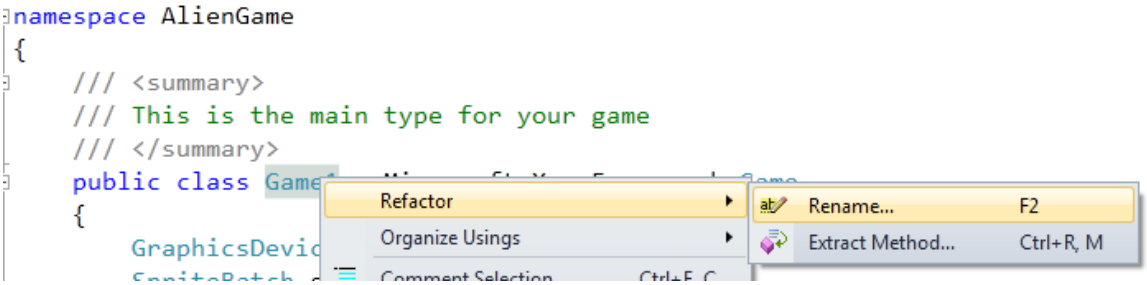


Figure 4

Renommer la classe principale du jeu

9. Dans la fenêtre qui s'affiche entrez AlienGame dans le champ **New name** et cliquez sur **OK**.

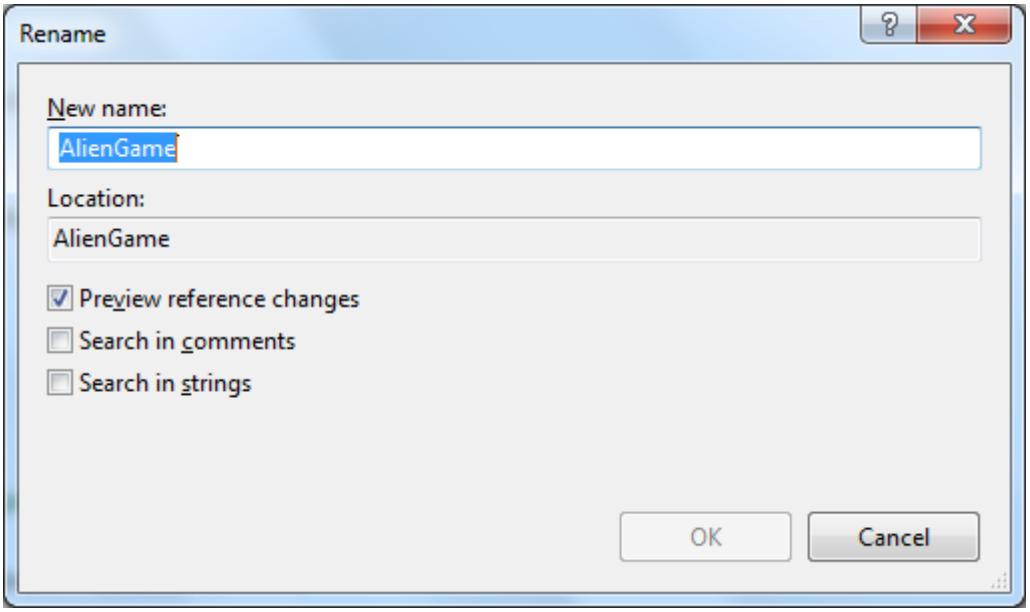


Figure 5

Donner un nom à la classe du jeu

10. Parcourez les changements que vous propose Visual Studio et cliquez sur le bouton **Apply** pour les appliquer.

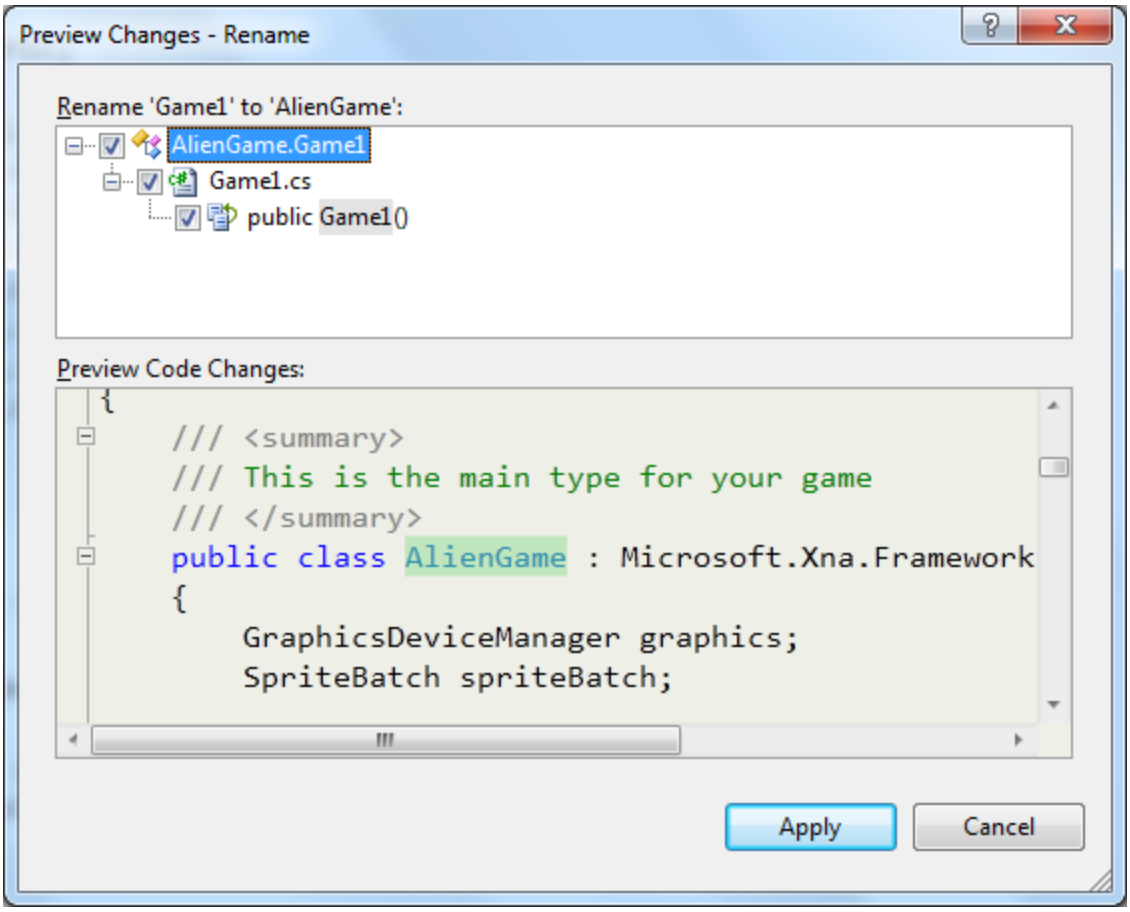


Figure 6

Application des changements dans la classe principale.

11. Renommez le fichier pour qu'il corresponde au nom de la classe. Dans le Solution Explorer, faites un clic droit sur le fichier **Game1.cs** et cliquez sur **Rename**. Nommez le fichier AlienGame.cs.

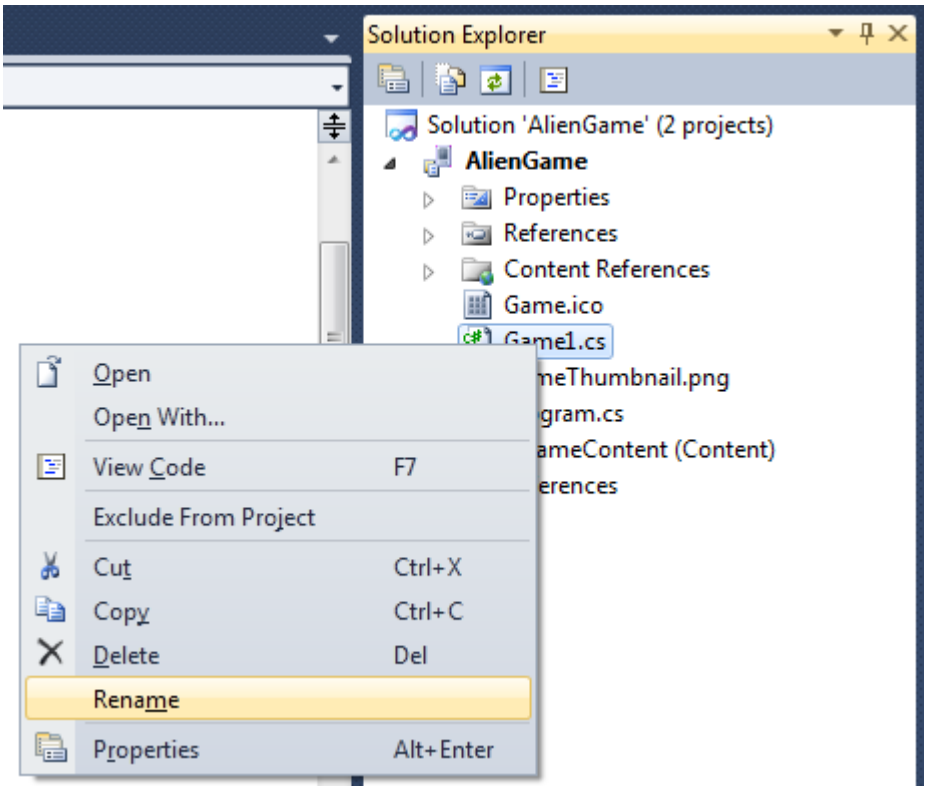


Figure 7

Renommer le fichier contenant la classe principale

12. Le fichier **GameThumbnail.png** contient l'icône permettant d'identifier le jeu dans l'écran de démarrage rapide de votre terminal Windows Phone. Vous pouvez double cliquez sur ce dernier dans le **Solution Explorer** pour ouvrir l'image dans une application externes installée sur votre machine, par exemple **Paint.exe** et ce dans le but de l'éditer. Vous réaliserez cette opération plus tard dans l'atelier.

> **Remarque:** Dans Visual Studio 2010, double cliquer sur le fichier dans le Solution Explorer ouvrira ce dernier dans un éditeur intégré à l'IDE.

Un jeu pour Windows développé avec le XNA Game Studio utilise des services fournis par la plateforme sur laquelle il s'exécute ou des librairies externes. Pour pouvoir utiliser ces fonctionnalités, il est nécessaire de référencer les librairies correspondantes.

13. Pour afficher les librairies référencées par le projet, déroulez le nœud **References** dans le **Solution Explorer** et regardez la liste qui s'affiche. Cette liste contient les librairies du Framework XNA ainsi que des librairies spécifiques à la plateforme Windows Phone.

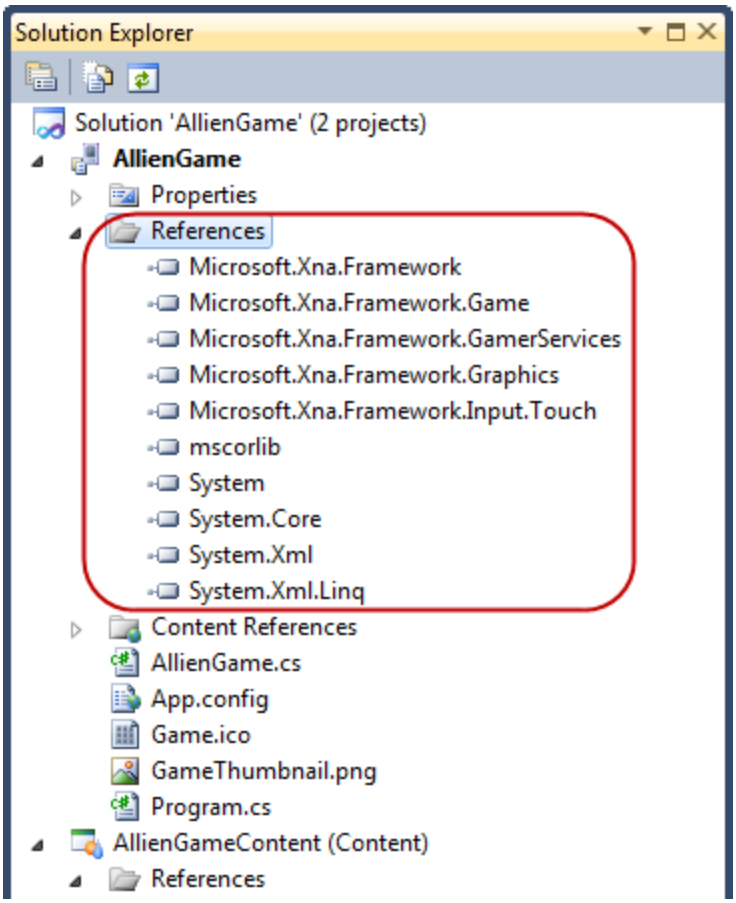


Figure 8

Liste des librairies référencées par le projet, dans le Solution Explorer

En l'état, l'application ne fait pas grand-chose, en revanche elle est prête à être exécutée pour la première fois. Les étapes qui suivent vont vous permettre de compiler l'application, déployer celle-ci dans l'émulateur Windows Phone ainsi que de l'exécuter afin de comprendre le cycle de vie du jeu.

14. Dans le menu**View**, choisissez l'entrée **Output** pour afficher la fenêtre de sortie (Output Window).
15. Dans le menu **Debug** cliquez sur **Build Solution** ou pressez **SHIFT + F6** pour compiler les projets présents dans la solution.

> **Visual Studio 2010:** Dans le menu **Build** cliquez sur **Build Solution** ou pressez **CTRL + SHIFT + B** pour compiler les projets présents dans la solution.

16. Observez alors les lignes qui s'affichent dans la fenêtre de sortie (**Output**) lors de la compilation de l'application :

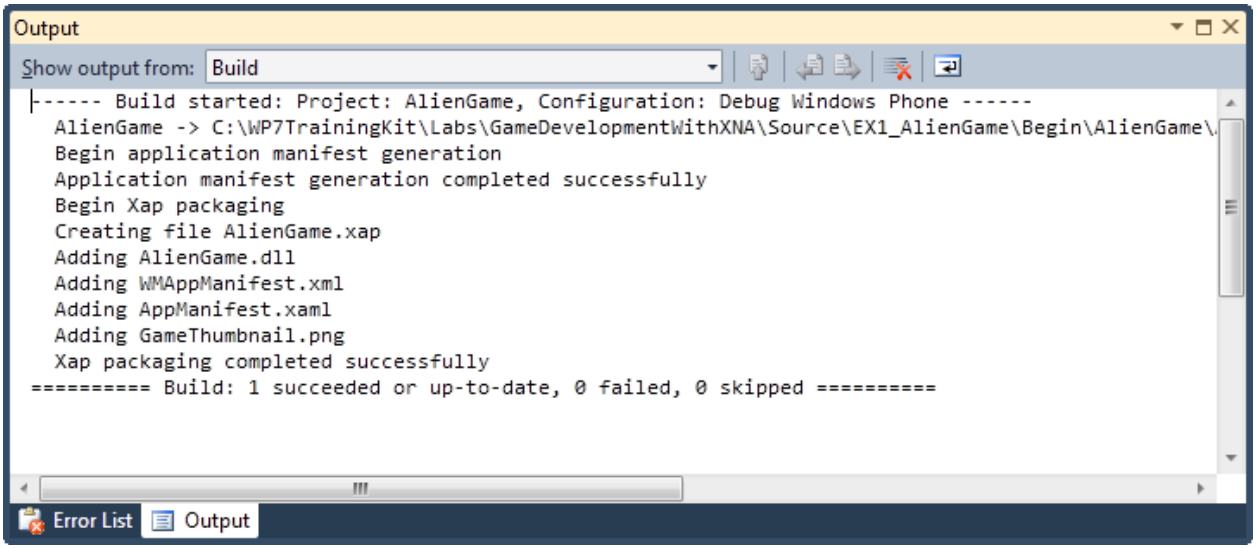


Figure 9

Compilation de l'application dans Visual Studio.

Normalement, vous ne devriez rencontrer aucune erreur lors de cette étape. Si des erreurs devaient se produire lors de la compilation, celles-ci serait affichées dans la fenêtre **Output**. Pour avoir des informations plus détaillées avec ces erreurs, vous pouvez utilizer la fenêtre **Error List**. Cette fenêtre affiche toutes les erreurs, mise en garde (Warnings) et messages retournés par le compilateur. Il est possible de filtrer les différents messages selon leur importance et de double cliquer sur l'un d'eux pour se rendre directement sur le bout de code incriminé.

17. Pour ouvrir la fenêtre d'erreur, rendez-vous dans le menu **View**, pointez **Other Windows** et cliquez sur **Error List**.

> **Visual Studio 2010:** Pour ouvrir la fenêtre d'erreur, ouvrez le menu **View** et cliquez sur **Error List**.

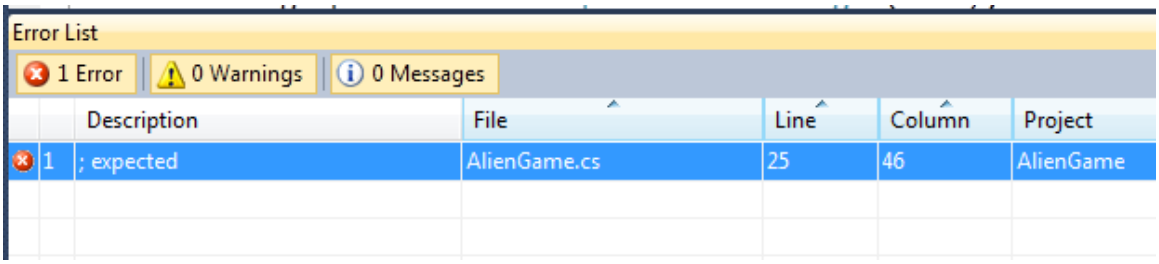


Figure 10

Fenêtre "Error List" affichant les erreurs survenues lors de la compilation.

18. Assurez vous que la cible pour le déploiement de l'application est bien l'émulateur Windows Phone. Pour se faire, verifies que l'entrée Windows Phone 7 Emulator et sélectionnée dans la

liste déroulante **Select Device** qui se trouve à côté du bouton **Start Debugging** dans la barre d'outils.

Figure 11

Choix de la cible de déploiement dans la barre d'outils.

Remarque: Lorsque vous utilisez Visual Studio pour déployer une application, il vous est possible de choisir un terminal Windows Phone reel comme cible de deployment

19. Appuyez sur **F5** pour démarrer l'application dans l'émulateur Windows Phone. La fenêtre de l'émulateur doit s'afficher et il y doit y avoir un petit temps de latence pendant que Visual Studio déploie l'application sur le périphérique virtuel. Une fois le déploiement effectué, l'écran de démarrage de Windows Phone 7 s'affiche et quelques instant après, votre application apparaît à l'écran.

    A ce stage de l'atelier, vous devriez voir apparaître un simple écran bleu pâle :

Figure 12

Exécution de l'application sur l'émulateur Windows Phone.

Tant que vous n'aurez pas développez la logique de l'application, vous ne pourrez rien faire de plus avec celle-ci.

20. Pressez **SHIFT + F5** ou cliquez sur le bouton **Stop** dans la barre d'outils pour arrêter la session de débogue et reprendre la main dans Visual Studio :

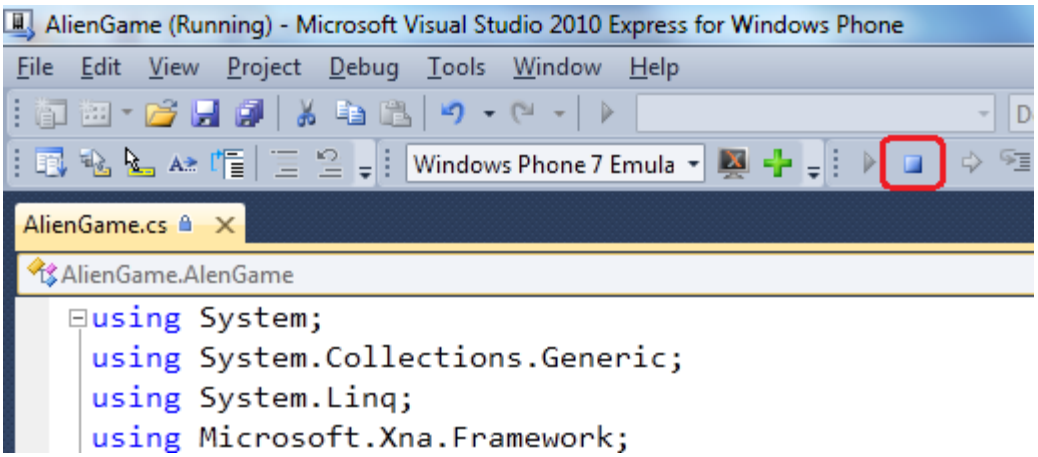


Figure 13

Terminer la session de débogue

> **Astuce:** Lorsque vous démarrez une session de débogue pour la première fois, le démarrage de l'application peut prendre un certain temps puisque l'émulateur Windows Phone doit s'initialiser. Pour gagner du temps entre les différents déploiements / sessions de débogue, ne fermez pas l'émulateur.

Exercice 2 – Utilisation de ressources dans un jeu XNA

La plupart des jeux font appelle à des images pour afficher le jeu à l'écran (décors, personnages, etc.), jouent des sons ou embarque d'autres ressources (vidéos, effets etc.). Cet atelier vous a été fournit avec un ensemble de ressource qui vous permettront de développer plus facilement et plus rapidement votre premier jeu. Pendant cet exercice, vous aller apprendre à ajouter ces ressources à votre jeu « AlienGame ».

De nombreux fichiers permettant la gestion des menus, des transitions entre les différents écrans du jeu vous ont également été fournis avec cet atelier. C'est dans cet exercice que vous les ajouterez au projet de jeu Windows Phone créé précédemment.

Vous êtes en train de developer le jeu. Celui-ci doit représenter un certain nombre d'éléments visuel à l'écran. Dans cet exercice, vous allez ajouter des ressources au projet ainsi qu'une partie de la logique de jeu pour la gestion des écrans (classe ScreenManager).

1. Fermez le projet actuellement ouvert dans Visual Studio. Dans l'explorateur Windows, copiez les deux fichiers suivant dans le repertoire "AlienGame":

- Game.ico
- PhoneGameThumb.png

2. Ouvrez Visual Studio 2010 et ouvrez le projet “AlienGame”.

   La plupart des jeux utilisent des dessins pour la creation des différents modèles, sprites, textures, effet, environnement de décors, animations etc. Tous ces éléments peuvent être créés de manières très différentes et peuvent être stockés dans différents formats. Ces éléments sont également succeptible d’être modifiés fréquemment tout au long du développement du jeu. Le Content Pipeline fournit est un outil pensé pour faciliter et accélerer la mise en place de ces éléments au sein d’un jeu développé avec le XNA Game Studio. Un dessinateur travaillant sur la représentation d’une voiture peut alors ajouter le fichier contenant sa création au sein du projet de jeu, lui donner un nom et choisir un outil d’importation (« importer ») ainsi qu’un « content processor » pour celui-ci. Dès lors, tout développeur qui souhaitera mettre la voiture en mouvement au sein du jeu pourra charger le modèle à partir de son nom, en appelant la méthode ContentManager.Load. Ceci permet d’améliorer la fluidité entre les équipes de création et les équipes de développement et surtout permet au dessinateur de se focuser sur ses modèles et au développeur de se focuser sur la logique du jeu sans perdre de temps pour transformer le travail de l’un à l’autre.

   Le XNA Content Pipeline est totalement integré à votre projet XNA Game Studio. Pour l’utiliser, il vous suffit d’ajouter des ressources au projet et de lancer une compilation de celles-ci. Les données sont alors importées et converties au format XNB (XNA Binary File) à l’aider d’un Content Importer. Le fichier XNB en question est généré différemment en fonction de la plateforme cible (Windows Phone, PC, Xbox). Les Content Importers sont implémentés dans des bibliothèques de classes fournies par le Framework XNA (il est possible de créer ses propres importers ou d’utiliser ceux créés par des tierces parties). Parmi les formats standards supportés par XNA, on a :

   - Format Autodesk FBX (.fbx)
   - Effets DirectX (.fx)
   - Description de police de caractères définie dans des fichiers.spritefont
   - Fichiers texture. Les formats suivant sont supportés: .bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, and .tga
   - Fichiers audio créés à l’aide de l’outil Microsoft Cross-Platform Audio Creation Tool (XACT), (au format .xap).

3. Cet atelier fournit un certain nombre de ressources multimédia telles que des polices de caractères, des sons et des images.Ajoutez les éléments suivants au projet AlienGameContent:
   - Toutes les polices de caractères du dossier \Assets\Media\Fonts
   - Toutes les images du dossier \Assets\Media\Images\Content
   - Tous les sons du dossier \Assets\Media\Sounds
4. Pour ajouter ces elements, faites un clic droit sur le nom du projet dans l’explorateur de solution et choisissez **Add** -> **Existing Items**:

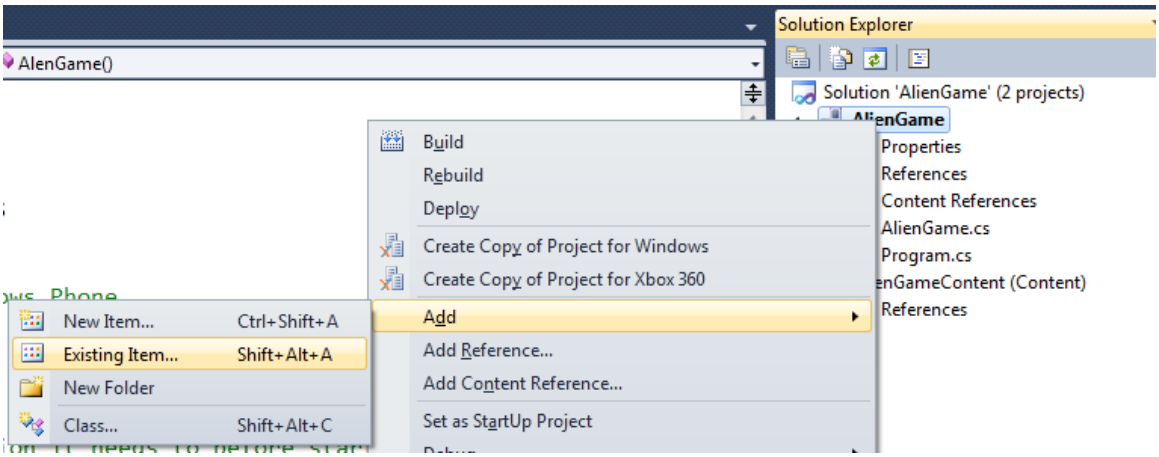


Figure 14

Ajouter des elements existants au projet

5. Rendez-vous aux différents emplacements énoncés ci-dessus et sélectionnez les fichiers. Dans certains cas, il est possible que vous ne voyiez pas tous le fichier. Vous pouvez changer le filtre dans la fenêtre de selection des fichiers pour modifier son affichage :

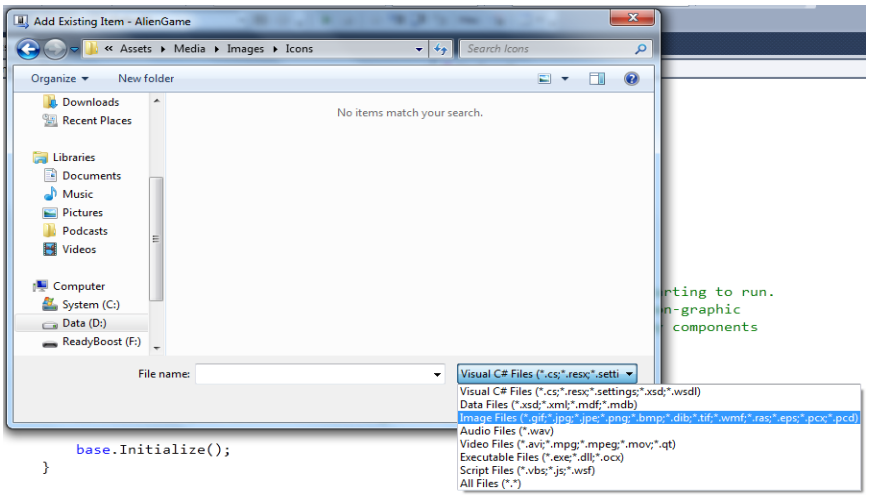
6. 

Figure 15

Modifier le filter sur les types de fichier dans le dialogue d'ajout d'éléments existants.

7. Une fois que vous avez ajoutez tous les fichiers, vous devriez obtenir l'affichage suivant pour le projet **AlienGameContent** dans le Solution Explorer :

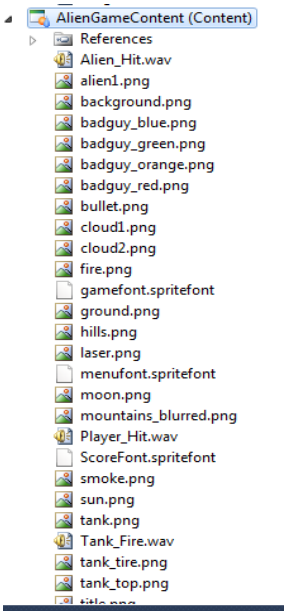


Figure 16

Structure du projet après ajout des différentes ressources

1. Ajoutez un dossier au projet en faisant un clic droit sur AlienGame et en sélectionnant l'entrée de menu Add -> New Folder

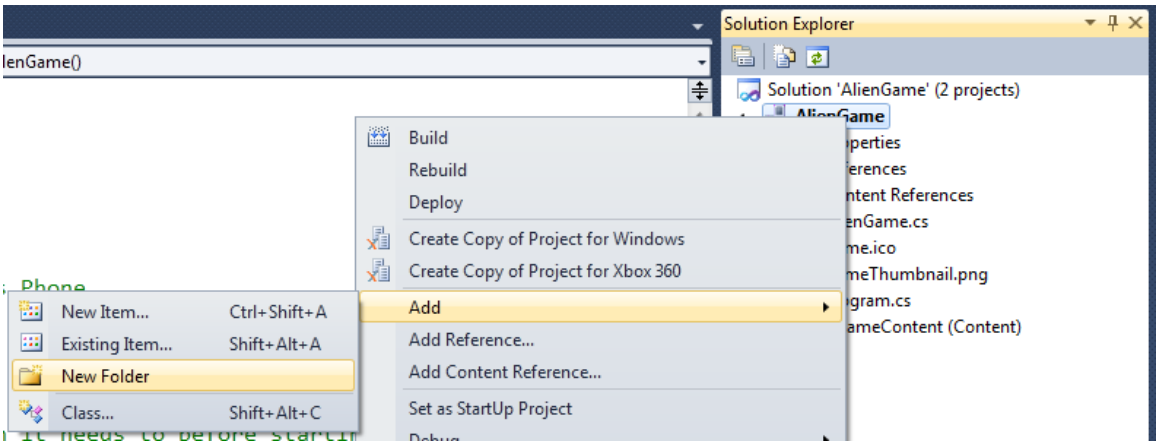


Figure 17

Ajout d'un nouveau dossier au projet

2. Appelez ce dossier **ScreenManager.**

C'est dans ce fichier que vous allez importer les différents fichiers de code source permettant de gérer la complexité de création des différents écrans du jeu, des menus ou encore des transitions entre ces différents éléments.

> **Astuce**: Ce code utilise l'approche standard en matière de creation de menu et écrans dans un jeu créé avec le XNA Game Studio.

3. Ajoutez tous les fichiers présents dans le repertoire \Assets\Code\ScreenManager dans le dossier que vous venez de créer.
4. Ajoutez également le fichier ParticleSystem.cs qui se trouve dans le répertoire \Assets\Code à la racine du projetAlienGame.
5. Votre projet devrait à present ressembler à l'écran ci-dessous:

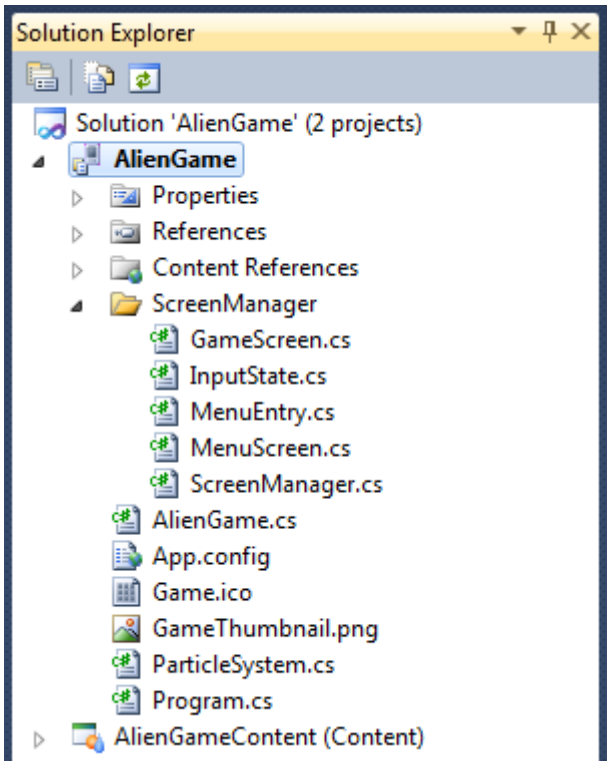


Figure 18

Structure du projet AlienGame après ajout des différents fichiers.

6. Ajoutez une nouvelle classe au projet AlienGame. Nommez-la "BackgroundScreen":

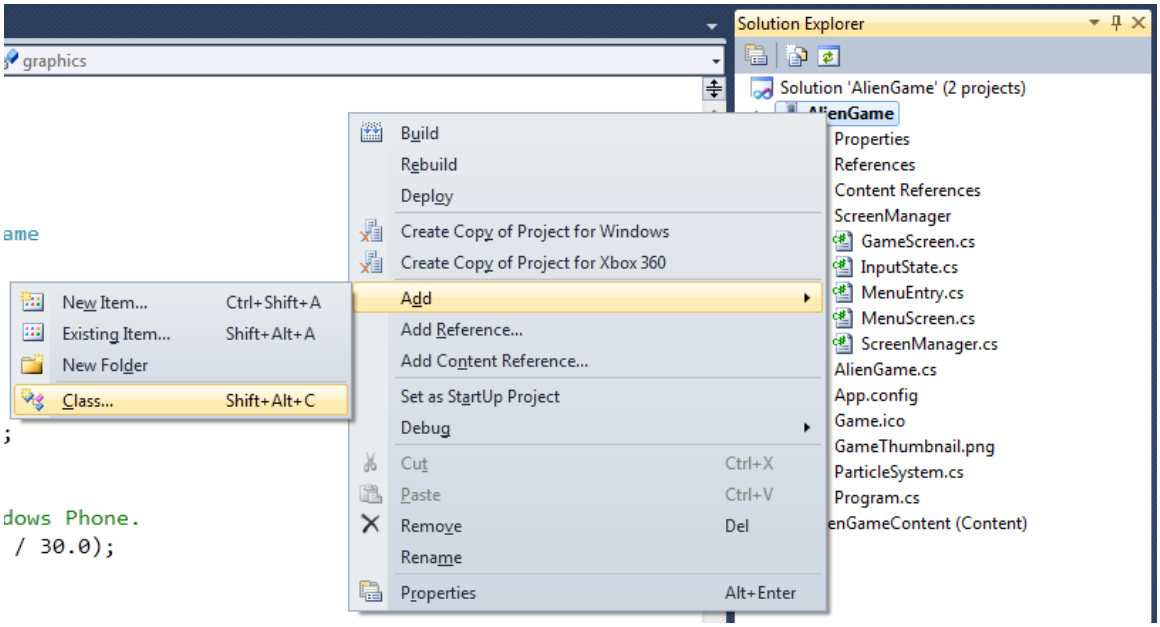


Figure 19

Ajout d'une nouvelle classe au projet

7. Ouvrez le fichier associé à cette classe et ajoutez-y les directives "using" suivantes :

(Code Snippet – *Développement de jeu pour WP7 avec le Framework XNA – directives using de la classe Background Screen*)

```
C#
using AlienGameSample;
using Microsoft.Xna.Framework.Graphics;
```

8. Cette classe derive de la classe GameScreen (définie dans les fichiers sources que vous avez ajoutés précédemment au dossier ScreenManager) :

```
C#
class BackgroundScreen : GameScreen
{
}
```

9. Ajoutez les variables de classes suivantes. Elles seront utilisées plus tard pour le chargement des ressources :

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – Variables de la classe BackgroundScreen*)

```
C#
Texture2D title;
Texture2D background;
```

10. Définissez à present le constructeur de la classe:

(Code Snippet – *Développement de jeu pour WP7 avec le Framework XNA –Constructeur de BackgroundScreen*)

```
C#
public BackgroundScreen()
{
   TransitionOnTime = TimeSpan.FromSeconds(0.0);
   TransitionOffTime = TimeSpan.FromSeconds(0.5);
}
```

11. La classe Game définit les différentes fonctionnalités qui ont été présentées lors de l'introduction de cet atelier : LoadContent, Update et Draw. Surchargez la méthode LoadContent :

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – Méthode LoadContent de la classe BackgroundScreen*)

```
C#
public override void LoadContent()
{
    title = ScreenManager.Game.Content.Load<Texture2D>("title");
    background = ScreenManager.Game.Content.Load<Texture2D>("background");
}
```

Cet extrait de code s'occupe de charger le contenu du jeu depuis les ressources ajoutées au jeu précédemment. Remarquez que les ressources sont chargées à l'aide de leurs noms.

12. Créez à present une classe LoadingScreen en répétant les étapes que vous avez apprises précédemment. L'écran qui sera affiché par cette classe sera celui affiché lors du chargement des ressources.
13. Ajoutez les directives using suivante à cette classe :

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – directives using de la classe LoadingScreen*)

```
C#
using AlienGameSample;
using System.Threading;
using Microsoft.Xna.Framework;
```

14. Cette nouvelle classe doit deriver de la classe GameScreen (comme pour la classe précédente). Ajoutez également un constructeur comme suivant à cette classe:

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – constructeur de la classe LoadingScreen*)

```
C#
public LoadingScreen()
{
   TransitionOnTime = TimeSpan.FromSeconds(0.0);
   TransitionOffTime = TimeSpan.FromSeconds(0.0);
}
```

15. Ajoutez une variable de classe de type Thread permettant de gérer le thread qui aura pour mission de charger les ressources du jeu:

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – variables de la classe LoadingScreen*)

```
C#
private Thread backgroundThread;
```

16. Ajoutez une method permettant de charger le contenu. Cette approche respecte les règles de développement standard d'un jeu XNA:

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – méthode BackgroundLoadContent de la classe LoadingScreen*)

```
C#
void BackgroundLoadContent()
{
   ScreenManager.Game.Content.Load<object>("alien_hit");
   ScreenManager.Game.Content.Load<object>("alien1");
   ScreenManager.Game.Content.Load<object>("background");
   ScreenManager.Game.Content.Load<object>("badguy_blue");
   ScreenManager.Game.Content.Load<object>("badguy_green");
   ScreenManager.Game.Content.Load<object>("badguy_orange");
   ScreenManager.Game.Content.Load<object>("badguy_red");
   ScreenManager.Game.Content.Load<object>("bullet");
   ScreenManager.Game.Content.Load<object>("cloud1");
   ScreenManager.Game.Content.Load<object>("cloud2");
   ScreenManager.Game.Content.Load<object>("fire");
   ScreenManager.Game.Content.Load<object>("gamefont");
   ScreenManager.Game.Content.Load<object>("ground");
   ScreenManager.Game.Content.Load<object>("hills");
   ScreenManager.Game.Content.Load<object>("laser");
   ScreenManager.Game.Content.Load<object>("menufont");
   ScreenManager.Game.Content.Load<object>("moon");
   ScreenManager.Game.Content.Load<object>("mountains_blurred");
   ScreenManager.Game.Content.Load<object>("player_hit");
   ScreenManager.Game.Content.Load<object>("scorefont");
   ScreenManager.Game.Content.Load<object>("smoke");
   ScreenManager.Game.Content.Load<object>("sun");
   ScreenManager.Game.Content.Load<object>("tank");
   ScreenManager.Game.Content.Load<object>("tank_fire");
   ScreenManager.Game.Content.Load<object>("tank_tire");
   ScreenManager.Game.Content.Load<object>("tank_top");
   ScreenManager.Game.Content.Load<object>("title");
   ScreenManager.Game.Content.Load<object>("titlefont");
}
```

17. Surchargez à present la method LoadContent de la classe mere de LoadingScreen et démarrez le chargement des ressources dans un nouveau thread afin que celles-ci soient chargées de manière asynchrone:

Remarque: Notre jeu étant très simple, les ressources seront chargées en une fraction de seconde. En revanche dans un jeu plus complexe, l'approche asynchrone permettrait d'afficher un indicateur de progression ou un écran d'attente.

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – méthdode LoadContent de la classe LoadingScreen*)

```
C#
public override void LoadContent()
{
   if (backgroundThread == null)
   {
      backgroundThread = new Thread(BackgroundLoadContent);
      backgroundThread.Start();
   }

   base.LoadContent();
}
```

18. Surchargez également la method Update de la classe mère GameScreen. Dans celle-ci, attendez que la méthode **LoadContent** se termine et ouvrez alors l'écran du menu principal (classe MainMenu, que vous ajouterez par la suite au projet):

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – method Update de la classe LoadingScreen*)

```
C#
public override void Update(GameTime gameTime, bool otherScreenHasFocus,
bool   coveredByOtherScreen)
{
    if (backgroundThread != null && backgroundThread.Join(10))
    {
        backgroundThread = null;
        this.ExitScreen();
        ScreenManager.AddScreen(new MainMenuScreen());
        ScreenManager.Game.ResetElapsedTime();
    }

    base.Update(gameTime, otherScreenHasFocus, coveredByOtherScreen);
}
```

19. Ajoutez une nouvelle classe au projet. Nommez-la **MainMenuScreen**.
20. Ajoutez-y les directives using suivantes :

(Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – directives using de la classe MainMenuScreen*)

```
C#
using AlienGameSample;
```

21. Dérivez cette nouvelle classe de la classe MenuScreen, elle aussi définie dans les fichiers de sources ajoutés au repertoire ScreenManager. Cette classe permet de faciliter l'affichage et la manipulation des menus.

22. Ajoutez un constructeur à la classe MainMenuScreen:

    (Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – Constructeur de la classe MainMenuScreen*)

```
C#
public MainMenuScreen() : base("Main")
{
   // Create our menu entries.
   MenuEntry startGameMenuEntry = new MenuEntry("START GAME");
   MenuEntry exitMenuEntry = new MenuEntry("QUIT");

   // Hook up menu event handlers.
   startGameMenuEntry.Selected += StartGameMenuEntrySelected;
   exitMenuEntry.Selected += OnCancel;

   // Add entries to the menu.
   MenuEntries.Add(startGameMenuEntry);
   MenuEntries.Add(exitMenuEntry);
}
```

23. Dans ce constructeur, vous venez de vous abonner à deux événements qui seront levés lorsque l'utilisateur sélectionnera l'un ou l'autre menu. Créez à presents les gestionnaires d'événements qui permettront de réaliser une action lorsque les événements seront levés :

    (Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – Gestionnaires d'événements de la classe MainMenuScreen*)

```
C#
void StartGameMenuEntrySelected(object sender, EventArgs e)
{

}

protected override void OnCancel()
{
   ScreenManager.Game.Exit();
}
```

24. Ouvrez le fichier AlienGame.cs.

25. Ajoutez-y les directives using suivantes :

    (Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – directives using de la classe AlienGame*)

```
C#
```

```
using AlienGameSample;
```

26. Ajoutez la variable de classe suivante :

    (Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – variables de la classe AlienGame*)

```
C#
ScreenManager screenManager;
```

27. Supprimez la variable spriteBatch créée par Visual Studio:

```
C#
SpriteBatch spriteBatch;
```

28. Supprimez également tout le code présent dans ce fichier à l'exception du constructeur de la classe.

29. Après avoir initialisé la classe principale du jeu, vous devez charger toutes les ressources du jeu et afficher le menu utilisateur à l'écran. L'écran de menu sera automatiquement affiché après le chargement des ressources. Il est également recommender de définir une résolution par défaut pour le périphérique qui exécutera le jeu. Pour réaliser toutes ces tâches, remplacez le code du constructeur existant par le code suivant:

    (Code Snippet - *Développement de jeu pour WP7 avec le Framework XNA – Constructeur de AlienGame*)

```
C#
public AlienGame()
{
    graphics = new GraphicsDeviceManager(this);

    //Set the Windows Phone screen resolution
    graphics.PreferredBackBufferWidth = 480;
    graphics.PreferredBackBufferHeight = 800;

    Content.RootDirectory = "Content";

    // Frame rate is 30 fps by default for Windows Phone.
    TargetElapsedTime = TimeSpan.FromSeconds(1 / 30.0);

    //Create a new instance of the Screen Manager
    screenManager = new ScreenManager(this);
    Components.Add(screenManager);

    //Add two new screens
    screenManager.AddScreen(new BackgroundScreen());
    screenManager.AddScreen(new LoadingScreen());
}
```

30. Compilez et exécutez l'application comme vous l'avez appris plus haut. Une fois que l'application aura chargé les ressources, le menu principal devrait s'afficher :

Figure 20
Menu principal de l'application

31. Terminez la session de débogue et retournez dans Visual Studio.

Dans cet exercice, vous avez appris à ajouter et manipuler des ressources au sein d'un jeu. Vous avez également créé un certain nombre d'écrans permettant de proposer une interface à l'utilisateur pendant le chargement des ressources ou encore d'afficher le menu principal.

Exercice 3 – La boucle de jeu

Dans cet exercice, vous allez vous ateler à implémenter les deux derniers éléments fondamentaux du jeu : la surcharge des méthodes Update et Draw.

1. Ouvrez le fichier BackgroundScreen.cs.
2. Surchargez la method **Update** de la classe mère comme le montre l'extrait de code ci-dessous:

   (Code Snippet – *Développement de jeux pour WP7 avec le Framework XNA – Méthode Update de la classe BackgroundScreen*)

```
C#
public override void Update(GameTime gameTime, bool otherScreenHasFocus,
                            bool coveredByOtherScreen)
{
   base.Update(gameTime, otherScreenHasFocus, false);
}
```

3. Surchargez également la méthode Draw.
4. Dans la méthode Draw, vous allez utiliser la classe SpriteBatch présente dans l'espace de noms Microsoft.Xna.Framewok.Graphics namespace pour dessiner les éléments du jeu à l'écran. Cela vous permet de dessiner un ensemble d'éléments tout en utilisant les mêmes paramètres. Modifiez le code de la méthode Draw comme suivant:

   (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode Draw de la classe BackgroundScreen*)

```
C#
public override void Draw(GameTime gameTime)
{
   SpriteBatch spriteBatch = ScreenManager.SpriteBatch;

   // Make the menu slide into place during transitions, using a
   // power curve to make things look more interesting (this makes
   // the movement slow down as it nears the end).
   float transitionOffset =           (float)Math.Pow(TransitionPosition,
2);

   spriteBatch.Begin();

   // Draw Background
   spriteBatch.Draw(background, new Vector2(0, 0),
            new Color(255, 255, 255, TransitionAlpha));

   // Draw Title
   spriteBatch.Draw(title, new Vector2(60, 55),
            new Color(255, 255, 255, TransitionAlpha));
```

```
    spriteBatch.End();
}
```

5. Compilez et exécutez l'application:

Figure 21

Application en execution après modification des méthodes Update et Draw

6. Arrêtez la session de débogue et retournez dans Visual Studio 2010
7. Ajoutez un nouvel écran à l'application : **GameplayScreen**
8. Dérivez cette nouvelle classe de la classe **GameScreen**.
9. Ajoutez les directives using suivante à la classe:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – directives using de la classe GameplayScreen*)

```
C#
using AlienGameSample;
using Microsoft.Xna.Framework;
```

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Audio;
using System.IO.IsolatedStorage;
using System.IO;
```

10. Ajoutez les variables de classe suivantes. Ces variables seront utilisées dans la suite pour l'implémentation de la logique du jeu, des entrées utilisateurs, du dessin, etc.

    (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA –Variables de la classe GameplayScreen*)

```
C#
//
// Game Play Members
//
Rectangle worldBounds;
bool gameOver;
int baseLevelKillCount;
int levelKillCount;
float alienSpawnTimer;
float alienSpawnRate;
float alienMaxAccuracy;
float alienSpeedMin;
float alienSpeedMax;
int alienScore;
int nextLife;
int hitStreak;
int highScore;
Random random;

//
// Rendering Members
//
Texture2D cloud1Texture;
Texture2D cloud2Texture;
Texture2D sunTexture;
Texture2D moonTexture;
Texture2D groundTexture;
Texture2D tankTexture;
Texture2D alienTexture;
Texture2D badguy_blue;
Texture2D badguy_red;
Texture2D badguy_green;
Texture2D badguy_orange;
Texture2D mountainsTexture;
Texture2D hillsTexture;
Texture2D bulletTexture;
Texture2D laserTexture;

SpriteFont scoreFont;
SpriteFont menuFont;
```

```
Vector2 cloud1Position;
Vector2 cloud2Position;
Vector2 sunPosition;

// Level changes, nighttime transitions, etc
float transitionFactor; // 0.0f == day, 1.0f == night
float transitionRate; // > 0.0f == day to night

ParticleSystem particles;

//
// Audio Members
//
SoundEffect alienFired;
SoundEffect alienDied;
SoundEffect playerFired;
SoundEffect playerDied;

//Screen dimension consts
const float screenHeight = 800.0f;
const float screenWidth = 480.0f;
const int leftOffset = 25;
const int topOffset = 50;
const int bottomOffset = 20;
```

11. Le constructeur de la classe GamePlay define la vitesse des transitions entre l'écran GameplayScreen et les différents écrans du jeu, la taille globale de l'environnement de jeu (le monde) ainsi que l'emplacement ou les différentes actions du jeu sont gérées. Ajoutez un constructeur à la classe GameplayScreen :

    (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Constructeur de la classe GameplayScreen*)

```
C#
public GameplayScreen()
{
   random = new Random();

   worldBounds = new Rectangle(0, 0, (int)screenWidth, (int)screenHeight);

   gameOver = true;

   TransitionOnTime = TimeSpan.FromSeconds(0.0);
   TransitionOffTime = TimeSpan.FromSeconds(0.0);
}
```

12. Créer les méthodes responsables du chargement et déchargement des ressources. Pour cela, surchargez les méthodes **LoadContent** et **UnloadContent** de la classe mère.
13. Ajoutez le code suivant à la méthode **LoadContent** :

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode LoadContent de la classe GameplayScreen*)

C#

```
public override void LoadContent()
{
    cloud1Texture = ScreenManager.Game.Content.Load<Texture2D>("cloud1");
    cloud2Texture = ScreenManager.Game.Content.Load<Texture2D>("cloud2");
    sunTexture = ScreenManager.Game.Content.Load<Texture2D>("sun");
    moonTexture = ScreenManager.Game.Content.Load<Texture2D>("moon");
    groundTexture = ScreenManager.Game.Content.Load<Texture2D>("ground");
    tankTexture = ScreenManager.Game.Content.Load<Texture2D>("tank");
    mountainsTexture =
ScreenManager.Game.Content.Load<Texture2D>("mountains_blurred");
    hillsTexture = ScreenManager.Game.Content.Load<Texture2D>("hills");
    alienTexture = ScreenManager.Game.Content.Load<Texture2D>("alien1");
    badguy_blue =
ScreenManager.Game.Content.Load<Texture2D>("badguy_blue");
    badguy_red = ScreenManager.Game.Content.Load<Texture2D>("badguy_red");
    badguy_green =
ScreenManager.Game.Content.Load<Texture2D>("badguy_green");
    badguy_orange =
ScreenManager.Game.Content.Load<Texture2D>("badguy_orange");
    bulletTexture = ScreenManager.Game.Content.Load<Texture2D>("bullet");
    laserTexture = ScreenManager.Game.Content.Load<Texture2D>("laser");
    alienFired = ScreenManager.Game.Content.Load<SoundEffect>("Tank_Fire");
    alienDied = ScreenManager.Game.Content.Load<SoundEffect>("Alien_Hit");
    playerFired =
ScreenManager.Game.Content.Load<SoundEffect>("Tank_Fire");
    playerDied =
ScreenManager.Game.Content.Load<SoundEffect>("Player_Hit");
    scoreFont = ScreenManager.Game.Content.Load<SpriteFont>("ScoreFont");
    menuFont = ScreenManager.Game.Content.Load<SpriteFont>("MenuFont");

    cloud1Position = new Vector2(224 - cloud1Texture.Width, 32);
    cloud2Position = new Vector2(64, 80);

    sunPosition = new Vector2(16, 16);

    particles = new ParticleSystem(ScreenManager.Game.Content,
ScreenManager.SpriteBatch);

    base.LoadContent();
}
```

14. Ajoutez le code suivant à la méthode **UnloadContent** :

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode UnloadContent de la classe GameplayScreen*)

C#

```
public override void UnloadContent()
{
    particles = null;

    base.UnloadContent();
}
```

15. Surchargez également la méthode **Update** :

> **Remarque**: Cette méthode sera modifiée plus tard dans cet atelier, lors de l'implémentation de la logique du jeu.

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA –Méthode Update de la classe GameplayScreen)*

C#

```
/// <summary>
/// Runs one frame of update for the game.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
public override void Update(GameTime gameTime,
      bool otherScreenHasFocus, bool coveredByOtherScreen)
{
   float elapsed = (float)gameTime.ElapsedGameTime.TotalSeconds;

   base.Update(gameTime, otherScreenHasFocus,      coveredByOtherScreen);
}
```

16. Surchargez la méthode Draw, responsable du dessin des éléments du monde de jeu environ 30 fois par seconde.

> **Astuce**: Le temps de jeu (GameTime) peut être utilisé pour calculer les déplacements des elements sur la surface de jeu:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode Draw de la classe GameplayScreen*)

C#

```
/// <summary>
/// Draw the game world, effects, and HUD
/// </summary>
/// <param name="gameTime">The elapsed time since last Draw</param>
public override void Draw(GameTime gameTime)
{
   float elapsedTime = (float)gameTime.ElapsedGameTime.TotalSeconds;

   ScreenManager.SpriteBatch.Begin();

   ScreenManager.SpriteBatch.End();
}
```

17. Ouvrez le fichier MainMenuScreen.cs et localisez la méthode StartGameMenuEntrySelected. Ajoutez alors une entrée pour afficher le GameplayScreen lorsque l'utilisateur sélectionne l'entrée de menu "START GAME":

    (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Ajout d'une entrée pour afficher GameplayScreen au niveau du menu du jeu*)

```
C#
ScreenManager.AddScreen(new GameplayScreen());
```

18. Compilez et exécutez l'application. Cliquez alors sur l'entrée de menu "START GAME" :

Figure 22

Exécution du jeu

> **Remarque**: Pour l'instant, l'écran de jeu est totalement vide. Vous allez ajouter le code necessaire à la logique et l'affichage du jeu par la suite.

19. Arrêtez la session de débogue et retournez dans Visual Studio 2010.

Dans cet exercice, vous avez créé la classe principale du jeu et surchargez plusieurs de ces méthodes.

Exercice 4 – Entrées utilisateur

Dans cet exercice, vous allez ajouter la gestion des entrées utilisateurs permettant de render le jeu jouable. Sur la plateforme Windows Phone 7, ces entrées utilisateurs peuvent se faire de plusieurs manière : via l'écran tactile ou à l'aide de l'accéléromètre.

> **Remarque**: L'accéléromètre n'est pas supporté par l'émulateur Windows Phone 7. Pour rendre le jeu jouable vous allez ajouter le support du clavier qui vous permettra de simuler l'utilisation de l'accéléromètre. Bien entendu, cela ne marchera pas sur un périphérique WP7 réel.

1. Ouvrez le fichier GameplayScreen.cs (si ce n'est pas déjà fait).
2. Ajoutez une reference à la librairie Microsoft.Device.Sensors.
3. Ajoutez une nouvelle directive using à la classe :

   (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Ajout d'une nouvelle directive using à la classe GameplayScreen*)

   C#

   ```
   using Microsoft.Xna.Framework.Input;
   using Microsoft.Xna.Framework.Input.Touch;
   using Microsoft.Devices.Sensors;
   ```

4. Ajoutez de nouvelles variables pour gérer les entrées de l'écran tactil et de l'accéléromètre:

   (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Ajout de nouvelles variables à la classe GameplayScreen*)

   C#

   ```
   //Input Members
   AccelerometerReading accelState;
   TouchCollection touchState;
   Accelerometer accelerometer;
   ```

5. Initialisez l'accéléromètre et abonnez-vous à ses différents événements. Pour se faire, ajoutez le code suivant au constructeur de la classe:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA –Initialisation de l'accéléromètre dans le constructeur de la classe*)

```
C#
if (accelerometer != null)
{
    if (accelerometer.State == SensorState.Ready)
    {
        accelerometer.CurrentValueChanged += (s, e) =>
        {
            accelState = e.SensorReading;
        };
        accelerometer.Start();
    }
}
```

6. Créez à présent une région « Input »:

```
C#
#region Input
#endregion
```

7. Dans cette region, ajoutez une surcharge de la méthode mere HandleInput:

Cette méthode sera responsable de la recuperation des entrées utilisateurs courante et sera utilisé dans la suite de cet atelier pour récupercuter les changements sur les valeurs des variables.

> **Remarque**: Dans l'émulateur, les clics de la souris seront interprétés comme des pressions sur l'écran ou des entrées clavier. Sur un périphérique Windows Phone réel, les entrées clavier ne se produiront jamais.

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode HandleInput de la classe GameplayScreen*)

```
C#
/// <summary>
/// Input helper method provided by GameScreen.  Packages up the various
input
/// values for ease of use.  Here it checks for pausing and handles
controlling
/// the player's tank.
/// </summary>
/// <param name="input">The state of the gamepads</param>
public override void HandleInput(InputState input)
{
   if (input == null)
      throw new ArgumentNullException("input");

   if (input.PauseGame)
   {
```

```
      if (gameOver == true)
         finishCurrentGame();
   }
   else
   {
      touchState = TouchPanel.GetState();
      bool buttonTouched = false;

      //interpret touch screen presses
      foreach (TouchLocation location in touchState)
      {
         switch (location.State)
         {
            case TouchLocationState.Pressed:
               buttonTouched = true;
               break;
            case TouchLocationState.Moved:
               break;
            case TouchLocationState.Released:
               break;
         }
      }

      float movement = 0.0f;
      if (Math.Abs(accelState.Acceleration.X) > 0.10f)
      {
         if (accelState.Acceleration.X > 0.0f)
            movement = 1.0f;
         else
            movement = -1.0f;
      }

      //TODO: Update player Velocity over X axis #1

      //This section handles tank movement.  We only allow one "movement"
action
      //to occur at once so that touchpad devices don't get double hits.
      KeyboardState keyState = Keyboard.GetState();

      if (input.CurrentGamePadStates[0].DPad.Left ==
       ButtonState.Pressed || keyState.IsKeyDown(Keys.Left))
      {
         //TODO: Update player velocity over X axis #2
      }
      else if (input.CurrentGamePadStates[0].DPad.Right ==
       ButtonState.Pressed || keyState.IsKeyDown(Keys.Right))
      {
         //TODO: Update player velocity over X axis #3
      }
      else
      {
         //TODO: Update player velocity over X axis #4
      }
```

```
      // B button, or pressing on the upper half of the pad or space on
keyboard or touching the touch panel fires the weapon.
      if (input.CurrentGamePadStates[0].IsButtonDown(Buttons.B) ||
input.CurrentGamePadStates[0].IsButtonDown(Buttons.A) ||
input.CurrentGamePadStates[0].ThumbSticks.Left.Y > 0.25f ||
keyState.IsKeyDown(Keys.Space) || buttonTouched)
      {
         if (!gameOver)
         {
            //TODO: Fire the bullet
         }
         else if (gameOver)
            finishCurrentGame();

      }
   }
}
```

8. Ajoutez une méthode utilisée pour terminer la partie en cours. Pour cela, ajoutez le code suivant à la classe GameplayScreen :

   (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode finishCurrentGame de la classe GameplayScreen*)

```
C#
private void finishCurrentGame()
{
   foreach (GameScreen screen in ScreenManager.GetScreens())
      screen.ExitScreen();

   ScreenManager.AddScreen(new BackgroundScreen());
   ScreenManager.AddScreen(new MainMenuScreen());
}
```

9. Compilez l'application.

   Dans cet exercice, vous avez ajouté la gestion des entrées utilisateurs au jeu AlienGame. Celles-ci seront utilisées dans la suite pour implémenter la logique du jeu.

Exercice 5 – Logique du jeu AlienGame

Dans cet exercice, vous allez implémenter la logique du jeu en tant que telle.

1. Dans le fichier GameplayScreen.cs, créez une nouvelle classe (en dessous de la definition de la classe GameplayScreen), comme le montre le code suivant :

   (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Classe Bullet définie dans le fichier GameplayScreen.cs*)

```C#
/// <summary>
/// Represents either an alien or player bullet
/// </summary>
public class Bullet
{
   public Vector2 Position;
   public Vector2 Velocity;
   public bool IsAlive;
}
```

2. Ajoutez également les deux classes suivantes au fichier :

   (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Classes Player et Alien, définies dans le fichier GameplayScreen.cs*)

```C#
/// <summary>
/// The player's state
/// </summary>
public class Player
{
   public Vector2 Position;
   public Vector2 Velocity;
   public float Width;
   public float Height;
   public bool IsAlive;
   public float FireTimer;
   public float RespawnTimer;
   public string Name;
   public Texture2D Picture;
   public int Score;
   public int Lives;
}

/// <summary>
/// Data for an alien.  The only difference between the ships
/// and the badguys are the texture used.
/// </summary>
public class Alien
{
   public Vector2 Position;
   public Texture2D Texture;
   public Vector2 Velocity;
   public float Width;
   public float Height;
   public int Score;
   public bool IsAlive;
   public float FireTimer;
   public float Accuracy;
   public int FireCount;
}
```

3. Ajoutez les variables suivantes à la classe GameplayScreen:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Ajout de variables à la classe GameplayScreen*)

C#

```
Player player;
List<Alien> aliens;
List<Bullet> alienBullets;
List<Bullet> playerBullets;
```

4. Initialisez ces variables dans le constructeur de la classe :

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Initialisation des variables Player et Alien dans le constructeur de GameplayScreen*)

C#

```
public GameplayScreen()
{
    player = new Player();
    playerBullets = new List<Bullet>();

    aliens = new List<Alien>();
    alienBullets = new List<Bullet>();

    if (accelerometer.State == SensorState.Ready)
```

5. Initialisez les proprieties Width et Height de la variable player dans la méthode LoadContent (après l'intialisation du système de particules et avant l'appel à base.LoadContent):

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Initialisation de la variable player dans la méthode LoadContent*)

C#

```
    player.Width = tankTexture.Width;
    player.Height = tankTexture.Height;
```

6. Les extraits de code suivant vous permettront d'ajouter la logique de jeu. Rendez-vous dans la méthode HandleInput et localisez la ligne suivante :

C#

```
//TODO: Update player Velocity over X axis #1
```

7. Ajoutez le code suivant sous cette ligne:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Gestion des mouvements du joueur dans la méthode HandleInput*)

```
C#
player.Velocity.X = movement;
```

8. Localisez la ligne suivante dans la méthode HandleInput, toujours :

```
C#
//TODO: Update player velocity over X axis #2
```

9. Ajoutez le code suivant sous cette ligne:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Gestion des mouvements du joueur dans la méthode HandleInput*)

```
C#
player.Velocity.X = -1.0f;
```

10. Localisez la ligne suivante :

```
C#
//TODO: Update player velocity over X axis #3
```

11. Ajoutez le code suivant sous cette ligne:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Gestion des mouvements du joueur dans la méthode Handleinput*)

```
C#
player.Velocity.X = 1.0f;
```

12. Localisez la ligne suivante :

```
C#
//TODO: Update player velocity over X axis #4
```

13. Ajoutez le code suivant sous celle-ci:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Gestion des mouvements de l'utilisateur dans la méthode HandleInput*)

```
C#
player.Velocity.X =
MathHelper.Min(input.CurrentGamePadStates[0].ThumbSticks.Left.X * 2.0f,
1.0f);
```

14. Localisez la ligne suivante :

```
C#
//TODO: Fire the bullet
```

15. Modifier l'instruction "if" comme suivant :

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Tir d'une balle dans la méthode HandleInput*)

```
C#
if (player.FireTimer <= 0.0f && player.IsAlive && !gameOver)
{
    Bullet bullet = CreatePlayerBullet();
    bullet.Position = new Vector2((int)(player.Position.X + player.Width /
2) - bulletTexture.Width / 2, player.Position.Y - 4);
    bullet.Velocity = new Vector2(0, -256.0f);
    player.FireTimer = 1.0f;

    particles.CreatePlayerFireSmoke(player);
    playerFired.Play();
}
else if (gameOver)
    finishCurrentGame();
```

16. Créez la méthode suivante dans la classe GameplayScreen :

Cette méthode sera responsable de la création d'une instance de la classe Bullet que vous avez définie ci-dessus. Cette méthode a été utilisée dans l'extrait de code ci-dessus.

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode CreatePlayerBullet de la classe GameplayScreen*)

```
C#
/// <summary>
/// Returns an instance of a usable player bullet.  Prefers reusing an ///
existing (dead)
/// bullet over creating a new instance.
/// </summary>
/// <returns>A bullet ready to place into the world.</returns>
Bullet CreatePlayerBullet()
{
   Bullet b = null;

   for (int i = 0; i < playerBullets.Count; ++i)
   {
      if (playerBullets[i].IsAlive == false)
      {
         b = playerBullets[i];
         break;
      }
```

```
    }

    if (b == null)
    {
        b = new Bullet();
        playerBullets.Add(b);
    }

    b.IsAlive = true;

    return b;
}
```

17. Modifier la méthode Update. Ajoutez le code suivant juste avant l'appel à "**base.Update(…)**":

    Ce bloque de code fournit la logique de jeu. Il déplace le joueur, appelle les méthodes pour mettre à jour les aliens et recalcul la position des balles qui ont été tirées.

    (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode Update de la classe GameplayScreen*)

```
C#
if (IsActive)
{
    // Move the player
    if (player.IsAlive == true)
    {
        player.Position += player.Velocity * 128.0f * elapsed;
        player.FireTimer -= elapsed;

        if (player.Position.X <= 0.0f)
            player.Position = new Vector2(0.0f, player.Position.Y);

        if (player.Position.X + player.Width >= worldBounds.Right)
            player.Position = new Vector2(worldBounds.Right - player.Width,
player.Position.Y);
    }

    Respawn(elapsed);

    UpdateAliens(elapsed);

    UpdateBullets(elapsed);

    CheckHits();

    if (player.IsAlive && player.Velocity.LengthSquared() > 0.0f)
        particles.CreatePlayerDust(player);

    particles.Update(elapsed);
```

```
}
```

18. Ajoutez les méthodes suivantes à la classe GameplayScreen:

> **Remarque**: L'extrait de code qui suit ajoute de nouvelles méthodes à la classe GameplayScreen. Vous trouverez ci-dessous une description de celles-ci.
>
> *Respawn*: Vérifie si le joueur est "mort" et si le jeu n'est pas terminé. Si ces deux conditions sont réussies, on attend que le timer "respawnTimer" se termine et on créé une nouvelle instance de la classe Player au milieu de l'écran.
>
> *UpdateBullets*: Vérifie et met à jour les positions des balles tirées par le joueur et les aliens.
>
> *UpdateAliens*: Déplace les aliens et calcule si ceux-ci doivent tirer une balle et dans quelle direction ils doivent le faire.
>
> *CheckHits*: Vérifie les collisions entre les balles et le joueur / les aliens. En plus de ça, cette méthode s'occuppe de mettre à jour le score, tuer les aliens ou encore terminer le jeu lorsqu'une collision se produit.

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthodes Respawn, UpdateBullets, UpdateAliens and CheckHits de la classe GameplayScreen*)

C#

```
/// <summary>
/// Handles respawning the player if we are playing a game and the player
is dead.
/// </summary>
/// <param name="elapsed">Time elapsed since Respawn was called
last.</param>
void Respawn(float elapsed)
{
    if (gameOver)
        return;

    if (!player.IsAlive)
    {
        player.RespawnTimer -= elapsed;
        if (player.RespawnTimer <= 0.0f)
        {
            // See if there are any bullets close...
            int left = worldBounds.Width / 2 - tankTexture.Width / 2 - 8;
            int right = worldBounds.Width / 2 + tankTexture.Width / 2 + 8;

            for (int i = 0; i < alienBullets.Count; ++i)
            {
                if (alienBullets[i].IsAlive == false)
                    continue;

                if (alienBullets[i].Position.X >= left ||
alienBullets[i].Position.X <= right)
```

```
                    return;
            }

            player.IsAlive = true;
            player.Position = new Vector2(worldBounds.Width / 2 -
player.Width / 2, worldBounds.Bottom - groundTexture.Height + 2 -
player.Height);
            player.Velocity = Vector2.Zero;
            player.Lives--;
        }
    }
}

/// <summary>
/// Moves all of the bullets (player and alien) and prunes "dead" bullets.
/// </summary>
/// <param name="elapsed"></param>
void UpdateBullets(float elapsed)
{
    for (int i = 0; i < playerBullets.Count; ++i)
    {
        if (playerBullets[i].IsAlive == false)
            continue;

        playerBullets[i].Position += playerBullets[i].Velocity * elapsed;

        if (playerBullets[i].Position.Y < -32)
        {
            playerBullets[i].IsAlive = false;
            hitStreak = 0;
        }
    }

    for (int i = 0; i < alienBullets.Count; ++i)
    {
        if (alienBullets[i].IsAlive == false)
            continue;

        alienBullets[i].Position += alienBullets[i].Velocity * elapsed;

        if (alienBullets[i].Position.Y > worldBounds.Height -
groundTexture.Height - laserTexture.Height)
            alienBullets[i].IsAlive = false;
    }
}

/// <summary>
/// Moves the aliens and performs their "thinking" by determining if they
/// should shoot and where.
/// </summary>
/// <param name="elapsed">The elapsed time since UpdateAliens was called
last.</param>
private void UpdateAliens(float elapsed)
```

```
{
    // See if it's time to spawn an alien;
    alienSpawnTimer -= elapsed;
    if (alienSpawnTimer <= 0.0f)
    {
        SpawnAlien();
        alienSpawnTimer += alienSpawnRate;
    }

    for (int i = 0; i < aliens.Count; ++i)
    {
        if (aliens[i].IsAlive == false)
            continue;

        aliens[i].Position += aliens[i].Velocity * elapsed;
        if ((aliens[i].Position.X < -aliens[i].Width - 64 &&
aliens[i].Velocity.X < 0.0f) ||
            (aliens[i].Position.X > worldBounds.Width + 64 &&
aliens[i].Velocity.X > 0.0f))
        {
            aliens[i].IsAlive = false;
            continue;
        }

        aliens[i].FireTimer -= elapsed;

        if (aliens[i].FireTimer <= 0.0f && aliens[i].FireCount > 0)
        {
            if (player.IsAlive)
            {
                Bullet bullet = CreateAlienBullet();
                bullet.Position.X = aliens[i].Position.X + aliens[i].Width
/ 2 - laserTexture.Width / 2;
                bullet.Position.Y = aliens[i].Position.Y +
aliens[i].Height;
                if ((float)random.NextDouble() <= aliens[i].Accuracy)
                {
                    bullet.Velocity = Vector2.Normalize(player.Position -
aliens[i].Position) * 64.0f;
                }
                else
                {
                    bullet.Velocity = new Vector2(-8.0f + 16.0f *
(float)random.NextDouble(), 64.0f);
                }

                alienFired.Play();
            }

            aliens[i].FireCount--;
        }
    }
}
```

```
/// <summary>
/// Performs all bullet and player/alien collision detection.  Also handles
game logic
/// when a hit occurs, such as killing something, adding score, ending the
game, etc.
/// </summary>
void CheckHits()
{
    if (gameOver)
        return;

    for (int i = 0; i < playerBullets.Count; ++i)
    {
        if (playerBullets[i].IsAlive == false)
            continue;

        for (int a = 0; a < aliens.Count; ++a)
        {
            if (aliens[a].IsAlive == false)
                continue;

            if ((playerBullets[i].Position.X >= aliens[a].Position.X &&
playerBullets[i].Position.X <= aliens[a].Position.X + aliens[a].Width) &&
(playerBullets[i].Position.Y >= aliens[a].Position.Y &&
playerBullets[i].Position.Y <= aliens[a].Position.Y + aliens[a].Height))
            {
                playerBullets[i].IsAlive = false;
                aliens[a].IsAlive = false;

                hitStreak++;

                player.Score += aliens[a].Score * (hitStreak / 5 + 1);

                if (player.Score > highScore)
                    highScore = player.Score;

                if (player.Score > nextLife)
                {
                    player.Lives++;
                    nextLife += nextLife;
                }

                levelKillCount--;
                if (levelKillCount <= 0)
                    AdvanceLevel();

                particles.CreateAlienExplosion(new
Vector2(aliens[a].Position.X + aliens[a].Width / 2, aliens[a].Position.Y +
aliens[a].Height / 2));

                alienDied.Play();
            }
```

```
        }
    }

    if (player.IsAlive == false)
        return;

    for (int i = 0; i < alienBullets.Count; ++i)
    {
        if (alienBullets[i].IsAlive == false)
            continue;

        if ((alienBullets[i].Position.X >= player.Position.X + 2 &&
alienBullets[i].Position.X <= player.Position.X + player.Width - 2) &&
(alienBullets[i].Position.Y >= player.Position.Y + 2 &&
alienBullets[i].Position.Y <= player.Position.Y + player.Height))
        {
            alienBullets[i].IsAlive = false;

            player.IsAlive = false;

            hitStreak = 0;

            player.RespawnTimer = 3.0f;
            particles.CreatePlayerExplosion(new Vector2(player.Position.X +
player.Width / 2, player.Position.Y + player.Height / 2));

            playerDied.Play();

            if (player.Lives <= 0)
            {
                gameOver = true;
            }
        }

    }
}

/// <summary>
/// Advances the difficulty of the game one level.
/// </summary>
void AdvanceLevel()
{
    baseLevelKillCount += 5;
    levelKillCount = baseLevelKillCount;
    alienScore += 25;
    alienSpawnRate -= 0.3f;
    alienMaxAccuracy += 0.1f;
    if (alienMaxAccuracy > 0.75f)
        alienMaxAccuracy = 0.75f;

    alienSpeedMin *= 1.35f;
    alienSpeedMax *= 1.35f;
```

```
    if (alienSpawnRate < 0.33f)
        alienSpawnRate = 0.33f;

    if (transitionFactor == 1.0f)
    {
        transitionRate = -0.5f;
    }
    else
    {
        transitionRate = 0.5f;
    }
}
```

19. Ajoutez le code suivant à la classe GameplayScreen:

> **Remarque**: Le code suivant ajoute également un certain nombre de méthodes dont les objectifs sont :
>
> *CreateAlienBullet*: Créer une instance de balle pouvant être utilisée par un alien pour tirer sur le joueur.
>
> *SpawnAlien*: Initialise une instance d'un nouvel alien, definie sa position initiale, sa vélocité, la couleur de sa texture…
>
> *CreateAlien*: Créée une nouvelle instance d'Alien et l'ajoute à la collection d'aliens

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthodes CreateAlienBullet, SpawnAlien and CreateAlien de la classe GameplayScreen*)

```
C#
/// <summary>
/// Returns an instance of a usable alien bullet.  Prefers reusing an
existing (dead)
/// bullet over creating a new instance.
/// </summary>
/// <returns>A bullet ready to place into the world.</returns>
Bullet CreateAlienBullet()
{
    Bullet b = null;

    for (int i = 0; i < alienBullets.Count; ++i)
    {
        if (alienBullets[i].IsAlive == false)
        {
            b = alienBullets[i];
            break;
        }
    }

    if (b == null)
    {
```

```
        b = new Bullet();
        alienBullets.Add(b);
    }

    b.IsAlive = true;

    return b;
}

/// <summary>
/// Creates an instance of an alien, sets the initial state, and places it
into the world.
/// </summary>
private void SpawnAlien()
{
    Alien newAlien = CreateAlien();

    if (random.Next(2) == 1)
    {
        newAlien.Position.X = -64.0f;
        newAlien.Velocity.X = random.Next((int)alienSpeedMin,
(int)alienSpeedMax);
    }
    else
    {
        newAlien.Position.X = worldBounds.Width + 32;
        newAlien.Velocity.X = -random.Next((int)alienSpeedMin,
(int)alienSpeedMax);
    }

    newAlien.Position.Y = 24.0f + 80.0f * (float)random.NextDouble();

    // Aliens
    if (transitionFactor > 0.0f)
    {
        switch (random.Next(4))
        {
            case 0:
                newAlien.Texture = badguy_blue;
                break;
            case 1:
                newAlien.Texture = badguy_red;
                break;
            case 2:
                newAlien.Texture = badguy_green;
                break;
            case 3:
                newAlien.Texture = badguy_orange;
                break;
        }
    }
    else
    {
```

```
            newAlien.Texture = alienTexture;
        }

        newAlien.Width = newAlien.Texture.Width;
        newAlien.Height = newAlien.Texture.Height;
        newAlien.IsAlive = true;
        newAlien.Score = alienScore;

        float duration = screenHeight / newAlien.Velocity.Length();

        newAlien.FireTimer = duration * (float)random.NextDouble();
        newAlien.FireCount = 1;

        newAlien.Accuracy = alienMaxAccuracy;
    }

    /// <summary>
    /// Returns an instance of a usable alien instance. Prefers reusing an
    existing (dead)
    /// alien over creating a new instance.
    /// </summary>
    /// <returns>An alien ready to place into the world.</returns>
    Alien CreateAlien()
    {
        Alien b = null;

        for (int i = 0; i < aliens.Count; ++i)
        {
            if (aliens[i].IsAlive == false)
            {
                b = aliens[i];
                break;
            }
        }

        if (b == null)
        {
            b = new Alien();
            aliens.Add(b);
        }

        b.IsAlive = true;

        return b;
    }
```

20. Rendez-vous dans la méthode Draw function et ajoutez le code mis en gras ci-dessous entre les appels de *Screen.SpriteBatch.Begin()* et *Screen.SpriteBatch.End()*:

    Cet ajout permettra à la méthode Draw d'appeler les différentes méthodes que vous allez créer ci-dessous et qui permettront de mettre à jour l'affichage après les calculs effectués par la méthode Update.

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Mise à jour de la méthode Draw de la classe GameplayScreen*)

```
C#
public override void Draw(GameTime gameTime)
{
    float elapsedTime = (float)gameTime.ElapsedGameTime.TotalSeconds;

    ScreenManager.SpriteBatch.Begin();

    DrawBackground(elapsedTime);
    DrawAliens();
    DrawPlayer();
    DrawBullets();
    particles.Draw();
    DrawForeground(elapsedTime);
    DrawHud();

    ScreenManager.SpriteBatch.End();
```

21. Ajoutez les méthodes suivantes dans la region “Draw” :

> **Note**: L’extrait de code suivant vous permet d’ajouter un certain nombre de méthodes responsables du dessin des elements de jeu à l’écran. Le détail de chacune de ces méthodes est indiqué ci-dessous :
>
> *DrawPlayer*: Dessine le joueur.
>
> *DrawAliens*: Dessine tous les aliens.
>
> *DrawBullets*: Dessine toutes les balles (celles du joueur et celles des aliens).
>
> *DrawForeground*: Dessine tous les nuages et les anime.
>
> *DrawBackground*: Dessine l’herbe, les montages, le soleil / la lune. S’occupe de la gestion des transitions entre le jour et la nuit.
>
> *DrawHud*: Dessine le score, le nombre de vies restantes et éventuellement l’information “Game over”.
>
> *DrawString*: Méthode générique permettant de dessigner du texte avec un effet d’ombre.

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthodes DrawPlayer, DrawAliens, DrawBullets, DrawForeground, DrawBackground, DrawHud and DrawString de la classe GameplayScreen*)

```
C#
/// <summary>
/// Draws the player's tank
/// </summary>
```

```
void DrawPlayer()
{
   if (!gameOver && player.IsAlive)
   {
      ScreenManager.SpriteBatch.Draw(tankTexture,
      player.Position, Color.White);
   }
}

/// <summary>
/// Draws all of the aliens.
/// </summary>
void DrawAliens()
{
   for (int i = 0; i < aliens.Count; ++i)
   {
      if (aliens[i].IsAlive)
      ScreenManager.SpriteBatch.Draw(aliens[i].Texture,                new
Rectangle((int)aliens[i].Position.X,
(int)aliens[i].Position.Y,
(int)aliens[i].Width,
(int)aliens[i].Height),
Color.White);
   }
}

/// <summary>
/// Draw both the player and alien bullets.
/// </summary>
private void DrawBullets()
{
   for (int i = 0; i < playerBullets.Count; ++i)
   {
      if (playerBullets[i].IsAlive)
         ScreenManager.SpriteBatch.Draw(bulletTexture,
      playerBullets[i].Position, Color.White);
   }

   for (int i = 0; i < alienBullets.Count; ++i)
   {
      if (alienBullets[i].IsAlive)
         ScreenManager.SpriteBatch.Draw(laserTexture,
             alienBullets[i].Position, Color.White);
   }
}

/// <summary>
/// Draw the foreground, which is basically the clouds. I think I had
planned on one point
/// having foreground grass that was drawn in front of the tank.
/// </summary>
/// <param name="elapsedTime">The elapsed time since last Draw</param>
private void DrawForeground(float elapsedTime)
```

```
{
   // Move the clouds.  Movement seems like an Update thing to do, but this
animations
   // have no impact over gameplay.
   cloud1Position += new Vector2(24.0f, 0.0f) * elapsedTime;
   if (cloud1Position.X > screenWidth)
      cloud1Position.X = -cloud1Texture.Width * 2.0f;

   cloud2Position += new Vector2(16.0f, 0.0f) * elapsedTime;
   if (cloud2Position.X > screenWidth)
      cloud2Position.X = -cloud1Texture.Width * 2.0f;

   ScreenManager.SpriteBatch.Draw(cloud1Texture,
      cloud1Position, Color.White);
   ScreenManager.SpriteBatch.Draw(cloud2Texture,    cloud2Position,
Color.White);
}

/// <summary>
/// Draw the grass, hills, mountains, and sun/moon. Handle transitioning
/// between day and night as well.
/// </summary>
/// <param name="elapsedTime">The elapsed time since last Draw</param>
private void DrawBackground(float elapsedTime)
{
   transitionFactor += transitionRate * elapsedTime;
   if (transitionFactor < 0.0f)
   {
      transitionFactor = 0.0f;
      transitionRate = 0.0f;
   }
   if (transitionFactor > 1.0f)
   {
      transitionFactor = 1.0f;
      transitionRate = 0.0f;
   }

   Vector3 day = Color.White.ToVector3();
   Vector3 night = new Color(80, 80, 180).ToVector3();
   Vector3 dayClear = Color.CornflowerBlue.ToVector3();
   Vector3 nightClear = night;

   Color clear = new Color(Vector3.Lerp(dayClear,
      nightClear, transitionFactor));
   Color tint = new Color(Vector3.Lerp(day, night,
      transitionFactor));

   // Clear the background, using the day/night color
   ScreenManager.Game.GraphicsDevice.Clear(clear);

   // Draw the mountains
   ScreenManager.SpriteBatch.Draw(mountainsTexture, new   Vector2(0,
screenHeight - mountainsTexture.Height),                  tint);
```

```
   // Draw the hills
   ScreenManager.SpriteBatch.Draw(hillsTexture, new       Vector2(0,
screenHeight - hillsTexture.Height), tint);

   // Draw the ground
   ScreenManager.SpriteBatch.Draw(groundTexture, new      Vector2(0,
screenHeight - groundTexture.Height),                      tint);

   // Draw the sun or moon (based on time)
   ScreenManager.SpriteBatch.Draw(sunTexture, sunPosition,
      new Color(255, 255, 255, (byte)(255.0f * (1.0f -
            transitionFactor))));
   ScreenManager.SpriteBatch.Draw(moonTexture, sunPosition,
      new Color(255, 255, 255, (byte)(255.0f *
            transitionFactor)));
}

/// <summary>
/// Draw the hud, which consists of the score elements and the GAME OVER
tag.
/// </summary>
void DrawHud()
{
   float scale = 2.0f;

   if (gameOver)
   {
      Vector2 size = menuFont.MeasureString("GAME OVER");
      DrawString(menuFont, "GAME OVER", new
Vector2(ScreenManager.Game.GraphicsDevice.Viewport.Width / 2 - size.X,
ScreenManager.Game.GraphicsDevice.Viewport.Height / 2 - size.Y / 2), new
Color(255, 64, 64), scale);

   }
   else
   {
      int bonus = 100 * (hitStreak / 5);
      string bonusString = (bonus > 0 ? " (" +
bonus.ToString(System.Globalization.CultureInfo.CurrentCulture) + "%)" :
"");
      // Score
      DrawString(scoreFont, "SCORE: " +
       player.Score.ToString(System.Globalization.CultureInfo
       .CurrentCulture) + bonusString, new    Vector2(leftOffset,
topOffset), Color.Yellow, scale);

     string text = "LIVES: " +
       player.Lives.ToString(System.Globalization.CultureInfo
       .CurrentCulture);
     Vector2 size = scoreFont.MeasureString(text);
     size *= scale;
```

```
        // Lives
        DrawString(scoreFont, text, new Vector2(screenWidth -
          leftOffset - (int)size.X, topOffset),
          Color.Yellow,       scale);

        DrawString(scoreFont, "LEVEL: " + (((baseLevelKillCount - 5) / 5) +
1).ToString(System.Globalization.CultureInfo.CurrentCulture), new
Vector2(leftOffset, screenHeight - bottomOffset), Color.Yellow, scale);

        text = "HIGH SCORE: " +
          highScore.ToString(System.Globalization.CultureInfo
.CurrentCulture);

        size = scoreFont.MeasureString(text);

        DrawString(scoreFont, text, new Vector2(screenWidth -
          leftOffset - (int)size.X * 2, screenHeight -
          bottomOffset), Color.Yellow, scale);
    }
}

/// <summary>
/// A simple helper to draw shadowed text.
/// </summary>
void DrawString(SpriteFont font, string text,
                    Vector2 position, Color color)
{
    ScreenManager.SpriteBatch.DrawString(font, text, new   Vector2(position.X
+ 1, position.Y + 1), Color.Black);
    ScreenManager.SpriteBatch.DrawString(font, text,       position, color);
}

/// <summary>
/// A simple helper to draw shadowed text.
/// </summary>
void DrawString(SpriteFont font, string text,
          Vector2 position, Color color, float fontScale)
{
    ScreenManager.SpriteBatch.DrawString(font, text, new   Vector2(position.X
+ 1, position.Y + 1), Color.Black,     0, new Vector2(0, font.LineSpacing /
2), fontScale,     SpriteEffects.None, 0);
    ScreenManager.SpriteBatch.DrawString(font, text,       position, color,
0, new Vector2(0, font.LineSpacing /   2), fontScale, SpriteEffects.None,
0);
}
```

22. Naviguez vers la méthode LoadContent et ajoutez le code suivant après l'appel de base.LoadContent():

```
C#

LoadHighscore();
```

```
Start();
```

23. Naviguez vers la méthode UnloadContent et ajoutez le code suivant avant l'instruction "particles = null;":

```
C#
SaveHighscore();
```

24. Créez une nouvelle region pour la gestion des meilleurs scores. Pour sauvegarder et charger des données depuis le système de fichier Windows Phone, vous devez utiliser l'isolated storage qui fournit pour chaque application un espace de stockage.Ajoutez le code suivant à la classe GameplayScreen pour ajouter cette fonctionnalité :

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthodes SaveHighscore et LoadHighscore de la classe GameplayScreen*)

```
C#
#region Highscore loading/saving logic
/// <summary>
/// Saves the current highscore to a text file. The StorageDevice was
selected during screen loading.
/// </summary>
private void SaveHighscore()
{
    using (IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication())
    {
        using (IsolatedStorageFileStream isfs = new
IsolatedStorageFileStream("highscores.txt", FileMode.Create, isf))
        {
            using (StreamWriter writer = new StreamWriter(isfs))
            {

writer.Write(highScore.ToString(System.Globalization.CultureInfo.InvariantC
ulture));
                writer.Flush();
                writer.Close();
            }
        }
    }
}

/// <summary>
/// Loads the high score from a text file.  The StorageDevice was selected
during the loading screen.
/// </summary>
private void LoadHighscore()
{
    using (IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication())
    {
```

```
        if (isf.FileExists("highscores.txt"))
        {
            using (IsolatedStorageFileStream isfs = new
IsolatedStorageFileStream("highscores.txt", FileMode.Open, isf))
            {
                using (StreamReader reader = new StreamReader(isfs))
                {
                    try
                    {
                        highScore = Int32.Parse(reader.ReadToEnd(),
System.Globalization.CultureInfo.InvariantCulture);
                    }
                    catch (FormatException)
                    {
                        highScore = 10000;
                    }
                    finally
                    {
                        if (reader != null)
                            reader.Close();
                    }
                }
            }
        }
    }
}
#endregion
```

25. Créez la méthode Start dans la classe GameplayScreen, permettant de démarrer une nouvelle partie:

    (Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthode Start de la classe GameplayScreen*)

```
C#
```

```
/// <summary>
/// Starts a new game session, setting all game states to initial values.
/// </summary>
void Start()
{
    if (gameOver)
    {
        player.Score = 0;
        player.Lives = 3;
        player.RespawnTimer = 0.0f;

        gameOver = false;

        aliens.Clear();
        alienBullets.Clear();
        playerBullets.Clear();
```

```
            Respawn(0.0f);
        }

        transitionRate = 0.0f;
        transitionFactor = 0.0f;
        levelKillCount = 5;
        baseLevelKillCount = 5;
        alienScore = 25;
        alienSpawnRate = 1.0f;

        alienMaxAccuracy = 0.25f;

        alienSpeedMin = 24.0f;
        alienSpeedMax = 32.0f;

        alienSpawnRate = 2.0f;
        alienSpawnTimer = alienSpawnRate;

        nextLife = 5000;
    }
```

26. Ouvrez le fichier ParticleSystem.cs.
27. Ajoutez la directive using suivante:

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – directive using dans la classe ParticleSystem*)

C#

```
using AlienGame;
```

28. Dans cette étape, vous allez ajouter deux méthodes qui vous permettront de créer les différents effets graphiques lors du déplacement du jouer ou encore du tir de balles. Ajoutez les méthodes suivantes à la classe :

(Code Snippet - *Développement de jeux pour WP7 avec le Framework XNA – Méthodes CreatePlayerDust et CreatePlayerFireSmoke de la classe ParticleSystem*)

C#

```
/// <summary>
/// Creates the mud/dust effect when the player moves.
/// </summary>
/// <param name="position">Where on the screen to create the effect.</param>
public void CreatePlayerDust(Player player)
{
    for (int i = 0; i < 2; ++i)
    {
        Particle p = CreateParticle();
        p.Texture = smoke;
        p.Color = new Color(125, 108, 43);
```

```
        p.Position.X = player.Position.X + player.Width *
(float)random.NextDouble();
        p.Position.Y = player.Position.Y + player.Height - 3.0f *
(float)random.NextDouble();
        p.Alpha = 1.0f;
        p.AlphaRate = -2.0f;
        p.Life = 0.5f;
        p.Rotation = 0.0f;
        p.RotationRate = -2.0f + 4.0f * (float)random.NextDouble();
        p.Scale = 0.25f;
        p.ScaleRate = 0.5f;
        p.Velocity.X = -4 + 8.0f * (float)random.NextDouble();
        p.Velocity.Y = -8 + 4.0f * (float)random.NextDouble();
    }
}

/// <summary>
/// Creates the effect for when the player fires a bullet.
/// </summary>
/// <param name="position">Where on the screen to create the effect.</param>
public void CreatePlayerFireSmoke(Player player)
{
    for (int i = 0; i < 8; ++i)
    {
        Particle p = CreateParticle();
        p.Texture = smoke;
        p.Color = Color.White;
        p.Position.X = player.Position.X + player.Width / 2;
        p.Position.Y = player.Position.Y;
        p.Alpha = 1.0f;
        p.AlphaRate = -1.0f;
        p.Life = 1.0f;
        p.Rotation = 0.0f;
        p.RotationRate = -2.0f + 4.0f * (float)random.NextDouble();
        p.Scale = 0.25f;
        p.ScaleRate = 0.25f;
        p.Velocity.X = -4 + 8.0f * (float)random.NextDouble();
        p.Velocity.Y = -16.0f + -32.0f * (float)random.NextDouble();
    }
}
```

29. Compilez et exécutez l’application. Sélectionnez “START GAME” dans le menu principal et jouez avec votre premier jeu !

Figure 23
Jeu terminé

Dans cet exercice, vous avez ajoutez la logique de jeu à AlienGame : calcul des mouvements du joueur et des aliens, gestion des collisions, mise à jour de l'écran…

Exercice 6 – Mixer Silverlight avec XNA

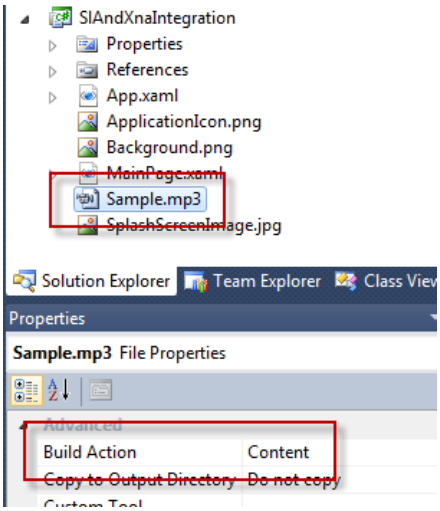
Il est possible d'intégrer des composants XNA directement au sein d'une application Silverlight et vice-versa. Ainsi, il est possible, dans son application Silverlight, de demander au framework XNA de

se charger de la lecture des fichiers audio (mais d'autres exemples, plus complexes, peuvent être envisagés).

1. Créer une nouvelle application Silverlight pour Windows Phone

2. Ajoutez une référence à Microsoft.Xna.Framework

3. Ajouter le fichier « Sample.mp3 » à votre projet et assurez-vous que sa propriété « Build Action » est bien positionnée à « Content » :



4. A présent, ajoutez le using suivant, qui vous permettra d'accéder aux classes de XNA :

C#

```
Microsoft.Xna.Framework.Media
```

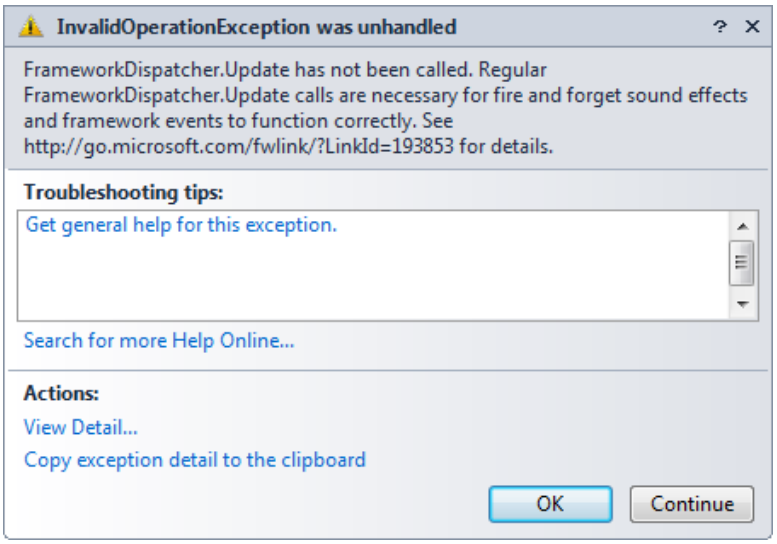
5. Dans le constructeur du fichier MainPage.xaml.cs, sous la méthode InitializeComponents, placez le code ci-dessous qui permet de déclarer un objet **Song**, qui représente un média à jouer, et qui déchence la lecture de ce média via un appel à la méthode Play du MediaPlayer, lecteur de flux audio de XNA :

C#

```
Song song = Song.FromUri("mySong", new Uri("Sample.mp3", UriKind.Relative));

MediaPlayer.Play(song);
```

6. A présent, si vous exécutez l'application, vous entendrez dans vos hauts-parleurs la musique se jouer mais il est également possible que vous rencontriez cette erreur :



7. Cette erreur survient si vous utilisez le Framework XNA dans une application qui n'implémente pas la classe Game. Dans ce cas, il est nécessaire d'appeler la méthode FrameworkDispatcher.Update à la main, pour gérer les messages qui sont dans la queue du Framework XNA. Pour cela, vous pouvez l'appeler à la main ou bien faire en sorte que la méthode soit appelée automatiquement (méthode idéale et plus simple, dans la majorité des cas).

8. Ajoutez une classe XnaAsyncDispatcher et copiez/collez le code suivant :

C#

```
public class XnaAsyncDispatcher : IApplicationService
{
    private readonly DispatcherTimer _frameworkDispatcherTimer;
    public XnaAsyncDispatcher(TimeSpan dispatchInterval)
    {
        FrameworkDispatcher.Update();
        _frameworkDispatcherTimer = new DispatcherTimer();
        _frameworkDispatcherTimer.Tick += FrameworkDispatcherTimer_Tick;
        _frameworkDispatcherTimer.Interval = dispatchInterval;
    }

    void IApplicationService.StartService(ApplicationServiceContext context)
    {
        _frameworkDispatcherTimer.Start();
    }

    void IApplicationService.StopService()
    {
        _frameworkDispatcherTimer.Stop();
    }

    private static void FrameworkDispatcherTimer_Tick(object sender, EventArgs
e)
    {
        FrameworkDispatcher.Update();
```

```
    }
}
```

9. Ouvrez le fichier App.xaml.cs et, à la fin du constructeur, ajoutez le code suivant :

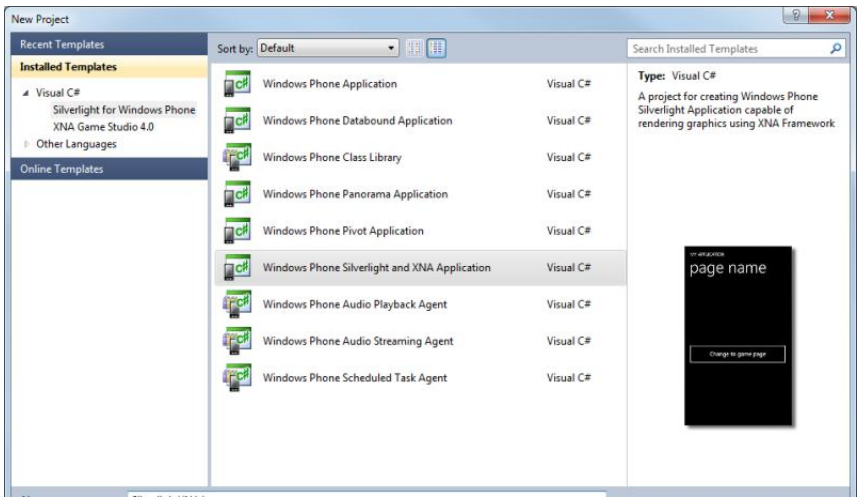
C#

```
this.ApplicationLifetimeObjects.Add(new
XnaAsyncDispatcher(TimeSpan.FromMilliseconds(50)));
```

10. Dès lors, si vous réèexécutez votre application, le son est correctement exécuté mais plus aucune exception n'est déclenchée !

Mais l'intégration entre Silverlight et XNA peut aller encore bien plus loin ! En effet, il est tout à fait possible de demander à Silverlight d'utiliser le Framework XNA pour effectuer tout ou partie de son rendu, comme nous allons le voir par la suite.

1. Créez une nouvelle application de type « Windows Phone Silverlight and XNA Application » depuis Visual Studio 2010 et nommez là « SlAndXnaApp » :



2. Sur le projet « SlAndXnaAppLibContent », faîtes un clic droit et sélectionnez « Add » => « Existing Item » et allez chercher le fichier « tank.png » qui est dans le répertoire « Assets\Images\Content »
3. Ouvrez le fichier GamePage.xaml.cs et déclarez la variable suivante :

C#

```
Texture2D tank;
```

4. Ensuite, allez jusqu'à la méthode OnNavigateTo et, après le commentaire « TODO : », ajoutez le code suivant :

C#

```
if (tank == null)
{
    tank = contentManager.Load<Texture2D>("tank");
}
```

5. Déclarez ensuite 2 variables de type Vector2, au niveau de la classe :

C#

```
Vector2 spritePosition;
Vector2 spriteSpeed = new Vector2(100.0f, 100.0f);
```

6. A présent, remplcez le contenu de la method OnUpdate par celui-ci :

C#

```
spritePosition += spriteSpeed * (float)e.ElapsedTime.TotalSeconds;

int MinX = 0;
int MinY = 0;
int MaxX = SharedGraphicsDeviceManager.Current.GraphicsDevice.Viewport.Width - tank.Width;
int MaxY = SharedGraphicsDeviceManager.Current.GraphicsDevice.Viewport.Height - tank.Height;

if (spritePosition.X > MaxX)
{
    spriteSpeed.X *= -1;
    spritePosition.X = MaxX;
}

else if (spritePosition.X < MinX)
{
    spriteSpeed.X *= -1;
    spritePosition.X = MinX;
}

if (spritePosition.Y > MaxY)
{
    spriteSpeed.Y *= -1;
    spritePosition.Y = MaxY;
}
else if (spritePosition.Y < MinY)
{
    spriteSpeed.Y *= -1;
    spritePosition.Y = MinY;
}
```

7. Et, pour la méthode OnDraw, remplacez le contenu par ce code :

C#

```
SharedGraphicsDeviceManager.Current.GraphicsDevice.Clear(Color.Black);

spriteBatch.Begin();
```

```
spriteBatch.Draw(tank, spritePosition, Color.White);

spriteBatch.End();
```

8. A present, nous allons rajouter un slider Silverlight qui va nous permettre de changer la couleur utilisée pour dessiner notre sprite. Ouvez le fichier GamePage.xaml et remplacez le commentaire par le code suivant :

C#

```
<Grid x:Name="LayoutRoot">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

    <Slider Minimum="0" Maximum="255" Value="255"
            Grid.Row="1"
            ValueChanged="OnValueChanged" />
</Grid>
```

9. Ouvrez le fichier GamePage.xaml.cs et ajoutez la méthode suivante, qui va permettre de modifier la valeur de la couleur utilisée pour dessiner notre tank :

C#

```
private double color = 255;

private void OnValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    color = e.NewValue;
}
```

10. Il ne reste plus qu'à utiliser cette couleur pour dessiner notre tank. Pour cela, remplacer le code de la méthode OnDraw par celui-ci :

C#

```
SharedGraphicsDeviceManager.Current.GraphicsDevice.Clear(Color.Black);

spriteBatch.Begin();

spriteBatch.Draw(tank, spritePosition, new Color((int) color, (int) color, (int) color));

spriteBatch.End();
```

11. Si à présent, vous exécutez l'application et que vous cliquez sur le bouton « Change to game page », vous remarquerez que le tank animé apparait bien mais pas le slider !

12. En effet, pour le moment, nous avons créé le slider mais n'oubliez pas que c'est le Framework XNA qui se charge du rendu des contrôles : il faut donc faire en sorte que celui-ci affiche aussi le slider, et cela va être possible grâce à la classe UIElementRenderer dont le rôle est d'afficher un contrôle Silverlight dans une texture 2D XNA.
13. Ainsi, déclarez le champ privé dans la classe GamePage :

C#

```
UIElementRenderer elementRenderer;
```

14. Dans le constructeur de la classe, abonnez-vous à l'évènement LayoutUpdated :

C#

```
this.LayoutUpdated += (o, e) => {
    if ((ActualWidth > 0) && (ActualHeight > 0))     {
        SharedGraphicsDeviceManager.Current.PreferredBackBufferWidth = (int)ActualWidth;
        SharedGraphicsDeviceManager.Current.PreferredBackBufferHeight = (int)ActualHeight;
    }

    if (null == elementRenderer)
    {
        elementRenderer = new UIElementRenderer(this, (int)ActualWidth, (int)ActualHeight);
    }
};
```

15. Enfin, dans la méthode OnDraw, avant la ligne « spriteBatch.Begin() », ajoutez le code suivant qui permet d'injecter le code XAML dans un buffer pouvant être accedé par la propriété Texture de la classe UIElementRenderer :

C#

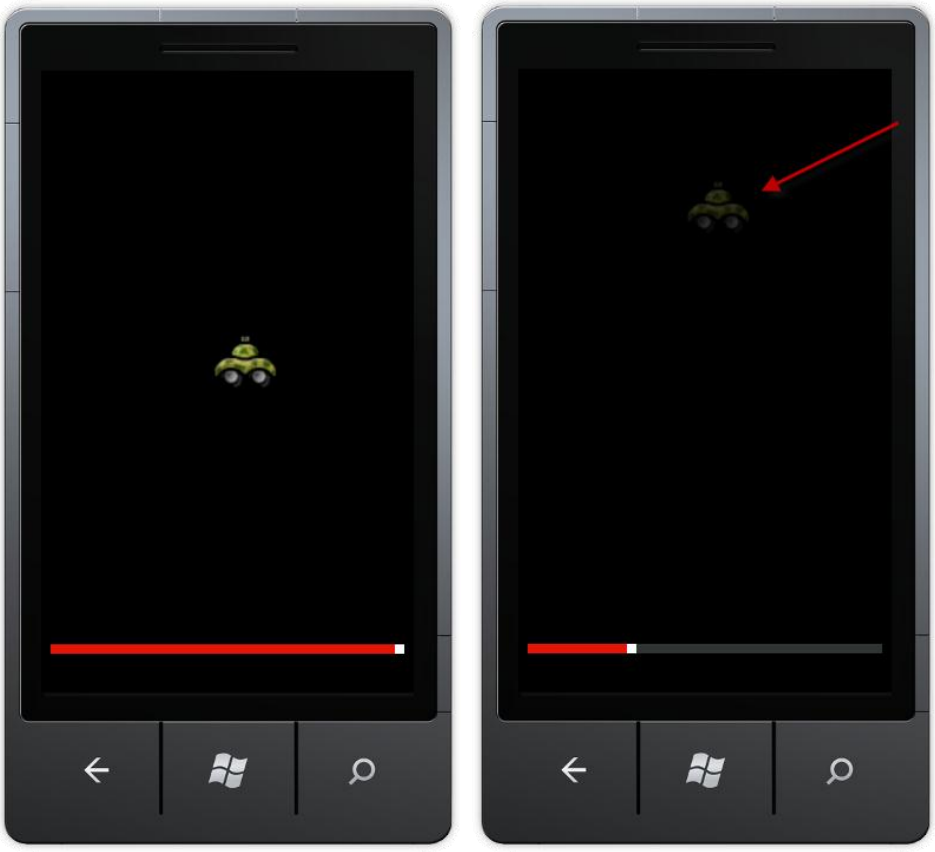
```
elementRenderer.Render();
```

16. Pour finir, il est nécessaire de demander au framework XNA d'afficher le composant UIElementRenderer, en injectant la ligne suivante, dans la méthode OnDraw, entre les appels à spriteBatch.Begin et spriteBatch.End :

C#

```
spriteBatch.Draw(elementRenderer.Texture, Vector2.Zero, Color.White);
```

17. Et c'est tout ! A present, si vous relancez l'application, vous pouvez constater que le slider est bien visible et, surtout, que si vous le manipuler, la couleur du sprite change en conséquence :



Résumé

Cet atelier a introduit les bases du développement d'application pour Windows Phone en utlisant le Framework XNA. Vous avez en effet créez un jeu à l'aide du XNA Game Studio pour Windows Phone. Pour cela, vous avez appris à gérer des ressources, prendre en compte les entrées utilisateur afin de mettre à jour l'état du jeu. Vous avez également appris à mettre en place la structure d'un jeu pour Windows Phone et à implémenter la logique du jeu AlienGame.

En réalisant cet atelier, vous avez également appris à prendre en main les différents outils pour créer et tester un jeu pour Windows Phone 7 : Microsoft Visual Studio 2010 pour Windows Phone et l'émulateur Windows Phone 7 !