Epreuve 5 du challenge WP7 : développez votre première application XNA

par Philippe Vialatte (Espace Perso) (Blog) Jérôme Lambert (Espace Perso) (Blog)

Date de publication : 05 août 2010

Dernière mise à jour :

Ce tutoriel va vous permettre de développer l'application XNA permettant de valider l'épreuve 5 du challenge Windows Phone 7.
Il vous guidera pas à pas, de façon à créer votre premier jeu avec XNA, un jeu de lettres basique.

I - Introduction....3
II - Récupération et description de la solution d'origine....3
III - Création des objets SolutionSquare et LetterSquare....5
IV - La classe de jeu....7
IV-A - Méthode LoadContent....8
IV-B - Méthode Update....10
IV-C - Méthode Draw....11
V - Conclusion....12

Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

I - Introduction

Le challenge Windows Phone 7 est l'occasion de découvrir les capacités de ce nouvel OS et de l'environnement de développement qui y est associé. Le but de ce tutoriel est de s'interesser aux capacités XNA de l'OS.

À la fin de ce tutoriel, vous saurez comment :

- Créer une application XNA pour Windows Phone 7
- Gérer les clics de la souris dans l'émulateur
- Tester la position d'éléments dans la surface de travail de l'émulateur

Nous allons écrire une application qui va nous fournir un mot dans le désordre que l'on devra ensuite réordonner pour valider l'épreuve.

XNA est un ensemble d'outils dont la finalité est de faciliter le développement de jeux sur les supports Windows, Xbox et Windows Phone. Il est à noter que les versions précédentes permettaient aussi le développement de jeux pour Zune, ce qui n'est plus le cas avec la version 4.0.
En particulier, on trouve, dans ces outils :

- XNA Game Studio 4.0, le kit de developpement que l'on va utiliser
- XNA Content Pipeline, dont le but est de standardiser la gestion des contenus
- Le Framework XNA en lui-même, qui va permettre de standardiser le développement des jeux entre toutes les plateformes concernées

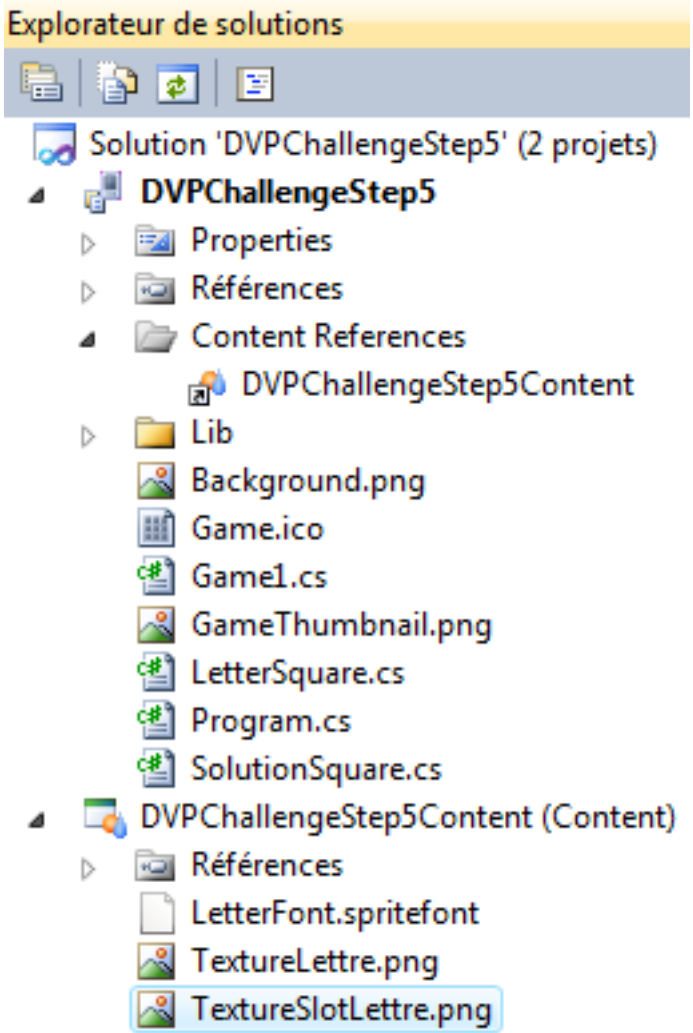
II - Récupération et description de la solution d'origine

Pour commencer au plus vite, et pour permettre la validation de l'épreuve, l'équipe du challenge vous convie à télécharger une version "à trous" de la solution du jeu.

Cette solution est disponible ici : **http://challenge-windowsphone7.developpez.com/Content/Code/Epreuve5.zip**

La solution comporte deux projets, DVPChallengeStep5 et DVPChallengeStep5Content. Le projet DVPChallengeStep5 est notre projet d'application XNA. Le projet DVPChallengeStep5Content, lui, est un projet de type XNA Content Pipeline Extensionn Library.

Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.



Le projet DVPChallengeStep5Content contient les éléments suivants :

- LetterFont.spritefont : ce fichier va nous permettre de gérer la police utilisée par les lettres de notre jeu
- TextureLettre.png : ce fichier nous permet de gérer la texture appliquée à l'arrière plan des lettres
- TextureSlotLettre.png : ce fichier nous permet de définir la texture appliquée à la zone de destination des lettres

Les fichiers de textures sont de simples images. Les fichiers de type SpriteFont sont des fichiers au format XML. Dans notre exemple, le fichier LetterFont.spritefont à le contenu suivant (commentaires enlevés pour mieux se rendre compte du contenu actif du fichier) :

```
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">
    <FontName>Segoe UI Mono</FontName>
    <Size>48</Size>
    <Spacing>0</Spacing>
    <UseKerning>true</UseKerning>
    <Style>Regular</Style>
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#233;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
```


Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

```
</XnaContent>
```

Le projet DVPChallengeStep5 contient les fichiers qui vont permettre de gérer les comportements du jeu. Trois fichiers vont nous interesser, LetterSquare.cs, SolutionSquare.cs, et Game1.cs.

LetterSquare va contenir les propriétés et les méthodes permettant de gérer les lettres que l'on va devoir remettre dans l'ordre.

SolutionSquare va contenir les propriétés permettant de gérer les zones dans lesquelles on va déposer les lettres.

Enfin, Game1 va nous permettre de gérer le chargement et le rendu du jeu.

III - Création des objets SolutionSquare et LetterSquare

On va commencer par l'objet le plus simple des deux, à savoir SolutionSquare.

SolutionSquare à besoin de gérer les informations suivantes :

- La position de la zone de solution, pour savoir où le positionner à l'écran
- La forme de la zone de solution, afin de déterminer la forme à dessiner
- Le centre du rectangle, de façon à pouvoir déterminer quand une lettre est déposée sur la zone de solution
- Enfin, un caractère permettant de stocker le caractère de la lettre qui à été déposée

SolutionSquare ne devant pas être déplacé, on va laisser le moteur de jeu s'occuper du rendu de ces objets. Cela va nous permettre de ne coder qu'un ensemble d'accesseurs, de la façon suivante :

```
public class SolutionSquare : ILetter
{
    public Vector2 Position;
    public Rectangle Rect;
    public Point Center;
    public char Letter { get; set; }
}
```

Les deux objets SolutionSquare et LetterSquare doivent implémenter l'interface ILetter pour que la validation se fasse comme prévu !

LetterSquare va contenir beaucoup plus de code. En effet, ces objets vont devoir être déplacés pour pouvoir former la solution, et vont de plus devoir gérer leur police, leur couleur de fond etc.
Le code pour cette classe comportera donc les propriétés suivantes :

```
public class LetterSquare : DrawableGameComponent, ILetter
{
    public char Letter { get; set; }
    public Vector2 Position;
    public Rectangle Rect;
    public TouchCollection TouchPoints;

    private SpriteFont Font;
    private Texture2D Background;
    private bool isSelected;
    private int myTouchPointId;
    private SpriteBatch spriteBatch;
    private GraphicsDevice graphicsDevice;
```


Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

Le constructeur de cette classe va simplement permettre d'initialiser les différentes valeurs de l'objet, de la façon suivante :

```
public LetterSquare(Game game, SpriteFont font, Texture2D background, Vector2 position)
    : base(game)
{
    this.graphicsDevice = game.GraphicsDevice;
    this.Font = font;
    this.Background = background;
    this.Position = position;
    this.Rect = new Rectangle((int)position.X, (int)position.Y, background.Width, background.Height);
}
```

Jusqu'ici, ca reste simple ...

Comme expliqué précedemment, ILetter doit être implémenté pour la validation. On voit, de plus, apparaitre la classe DrawableGameComponent, que LetterSquare va implémenter. Hériter de DrawableGameComponent permet que les objets LetterSquare, une fois ajouté à la liste des composants du jeu, soient automatiquement mis à jour et redessiné par le jeu les contenant, sans avoir à leur redemander explicitement de se redessiner.

On va tout de même, pour cela, remplacer les méthodes Update et Draw de DrawableGameComponent, de la façon suivante :

```
public override void Update(GameTime gameTime)
{
    foreach (var touchPoint in TouchPoints)
    {
        switch (touchPoint.State)
        {
            case TouchLocationState.Pressed:
                if (this.Rect.Contains((int)touchPoint.Position.X, (int)touchPoint.Position.Y))
                {
                    this.isSelected = true;
                    this.myTouchPointId = touchPoint.Id;
                }
                break;
            case TouchLocationState.Moved:
                if (this.isSelected && touchPoint.Id == myTouchPointId)
                {
                    this.Position.X = touchPoint.Position.X - this.Rect.Width / 2;
                    this.Position.Y = touchPoint.Position.Y - this.Rect.Height / 2;
                    this.Rect.X = (int)this.Position.X;
                    this.Rect.Y = (int)this.Position.Y;
                }
                break;
            case TouchLocationState.Released:
                if (this.isSelected && touchPoint.Id == myTouchPointId)
                {
                    this.isSelected = false;
                    myTouchPointId = 0;
                }
                break;
            default:
                break;
        }
    }
    base.Update(gameTime);
}
```

La méthode Update va permettre de gérer les pressions de l'utilisateur sur l'écran (en l'occurences, les clics avec la souris). En effet, le jeu, enverra depuis sa méthode Update les differents point touchés par l'utilisateur aux différentes lettres présentes à l'écran. Charge ensuite aux lettres de déterminer si elles sont concernées par un de ces points.

Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

```
public override void Draw(GameTime gameTime)
{
    spriteBatch = new SpriteBatch(this.graphicsDevice);
    spriteBatch.Begin();
    spriteBatch.Draw(this.Background, this.Rect, Color.White);
    spriteBatch.DrawString(this.Font, this.Letter.ToString(), new Vector2(Position.X + 20,
 Position.Y + 20), Color.White);
    spriteBatch.End();

    base.Draw(gameTime);
```

La méthode Draw va, elle, permettre de redessiner une lettre lorsque nécessaire. Pour cela, on va d'abord créer un objet SpriteBatch, qui va nous permettre de gérer un lot de Sprite d'une seule traite (dans le but d'améliorer les performances). On va ensuite ajouter au SpriteBatch les éléments représentatifs de notre LetterSquare, à savoir la lettre et le carré qui l'entourent. A noter, le carré prendra comme couleur celle définie dans la texture contenue dans Background, car on a défini comme couleur de teinte la couleur blanche. En effet, pour obtenir la couleur finale d'un pixel, le moteur de rendu fera un ET logique entre la couleur du pixel dans la texture, et la couleur passée en argument.

Nos objets étant prêts, on va maintenant s'intéresser à la classe de jeu.

IV - La classe de jeu

La classe de jeu ne contient pour l'instant qu'un squelette maigrelet, qui est le suivant :

```
    public class Game1 : BaseGame
    {
        public Game1()
        {
            GraphicsDeviceManager graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        protected override void LoadContent()
        {
            // Création des textures et de la fonte

            // Création des objets LetterSquare et SolutionSquare
   // le bon nombre de cases à créer venant de la dll de validation (NumberOfSquares)
            for (int i = 0; i < this.NumberOfSquares(); i++)
            {
            }
            //Chargement des lettre depuis la classe externe
            LoadLetters();
        }

        protected override void Update(GameTime gameTime)
        {
            // Allows the game to exit
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();

            // Récupération des TouchPoints, pour les envoyer aux objets LetterSquare

   // Si l'objet LetterSquare est assez proche de la case de solution, le caser sur la case en question

   // Rappel de la méthode de base de mise à jour
            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            // Vérification de la solution proposée
            TestForSolution("login", "password");

   // Dessiner toutes les cases de solution (les LetterSquare se redessineront seuls)
```


Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

```
    // Si la solution est trouvée, afficher le message "C'est gagné !"

    // On rappelle la méthode de base pour dessiner automqtiauement les autres objets
            base.Draw(gameTime);
        }
     }
}
```

Si on lance immédiatement le projet, le résultat est un peu décevant (mais logique)

On va donc commencer par charger les lettres.

IV-A - Méthode LoadContent

La méthode LoadContent sera appelée automatiquement lorsque l'on lance l'application. On va commencer par créer les structures dont on a besoin :

```
// Création des textures et de la fonte
LetterBackground = Content.Load<Texture2D>("TextureLettre");
LetterSlotBackground = Content.Load<Texture2D>("TextureSlotLettre");
LetterFont = Content.Load<SpriteFont>("LetterFont");
```


Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

A noter, ces trois variables sont déclarées, comme de nombreuses autres, dans la classe BaseGame, qui fait partie de la dll cryptée de validation.

Ici, on va simplement charger nos deux structures, ainsi que la police que l'on va utiliser.

On va ensuite créer le nombre adéquat d'objets LetterSquare et solutionSquare

```
// Création des objets LetterSquare et SolutionSquare
// le bon nombre de cases à créer venant de la dll de validation (NumberOfSquares)
for (int i = 0; i < this.NumberOfSquares(); i++)
{
    // création d'une Lettre
    LetterSquare lstmp = new LetterSquare(this, LetterFont, LetterBackground, new Vector2(106 *
 i, 100));
    this.Components.Add(lstmp);
    Letters.Add(lstmp);


  // Création d'une case de solution. Inutile de l'ajouter à la liste des composants, elle sera gérée en manuel
    SolutionSquare sstmp = new SolutionSquare();
    sstmp.Position = new Vector2(106 * i, 300);
    sstmp.Rect = new Rectangle((int)sstmp.Position.X, (int)sstmp.Position.Y,
 LetterSlotBackground.Width, LetterSlotBackground.Height);
    sstmp.Center = new Point((int)sstmp.Position.X + LetterSlotBackground.Width / 2,
 (int)sstmp.Position.Y + LetterSlotBackground.Height / 2);
    SolutionSlots.Add(sstmp);
}
```

La seule chose que l'on n'ait pas modifée est l'appel à la fonction LoadLetters, qui va charger dans les LetterSquares les caractères à ré-ordonner.

Lorsque l'on va re-lancer l'application, on va voir l'image suivante (lettres floutées pour décourager les tricheurs ;))

Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

Les lettres ne peuvent pas être bougées, on va donc passer maintenant à la gestion des événements.

IV-B - Méthode Update

La méthode Update va permettre de gérer les mises à jour des différents objets et événements durant les boucles du jeu.

Comme vu dans les commentaires, on va commencer par récupérer tous les points actifs (TouchPoints) de la surface du jeu, et on va les transférer à nos lettres.

```
// Récupération des TouchPoints, pour les envoyer aux objets LetterSquare
TouchCollection Touches = TouchPanel.GetState();

foreach (LetterSquare gc in Letters)
{
    gc.TouchPoints = Touches;
}
```

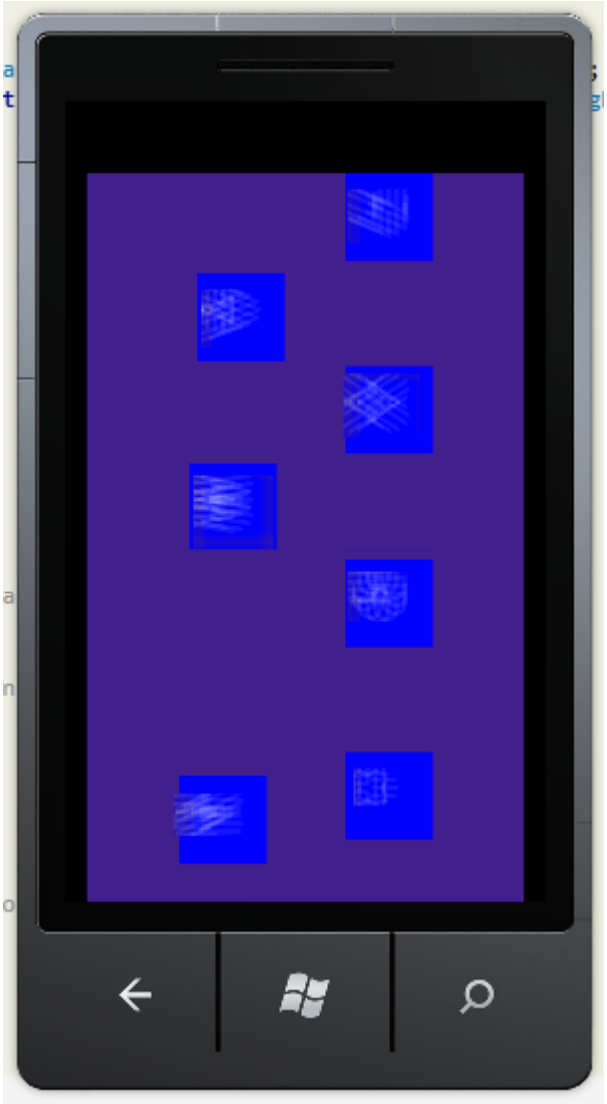
On va, de plus, s'assurer que si les lettres sont suffisament proches d'une zone de solution, elles vont se placer dessus automatiquement (pour éviter au joueur de devoir placer les éléments au pixel près).

```
foreach (LetterSquare letter in Letters)
```


Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

```
{
    foreach (SolutionSquare solution in SolutionSlots)
    {
        if (letter.Rect.Contains(solution.Center))
        {
            letter.Rect = solution.Rect;
            letter.Position = solution.Position;
            solution.Letter = letter.Letter;
        }
    }
}
```

Si on lance notre jeu, on peut désormais déplacer les lettres à la souris.



Il ne nous reste plus maintenant qu'à dessiner les carrés entourant nos cases de solution, et à avertir l'utilisateur lorsqu'il a gagné (ou lui renvoyer un message d'erreur si la validation échoue...).

IV-C - Méthode Draw

La méthode Draw va être principalement utilisée pour dessiner les rectangles, et pour informer l'utilisateur de la validation de l'épreuve, ou d'un échec de validation. On va aussi redessiner le fond de l'écran en noir. Cette métode étant relativement simple, elle se passe de d'avantage de commentaires

Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.

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

var spriteBatch = new SpriteBatch(GraphicsDevice);
spriteBatch.Begin();

foreach (SolutionSquare slot in SolutionSlots)
    spriteBatch.Draw(LetterSlotBackground, slot.Rect, Color.White);

switch (myGameState)
{
    case GameState.InProgress:
        break;
    case GameState.SolutionFound:
    spriteBatch.DrawString(LetterFont, "C'est gagné !", new Vector2(200, 100), Color.White);
        break;
    case GameState.Error:
    spriteBatch.DrawString(LetterFont, "Erreur de validation !", new Vector2(200, 100), Color.White);
        break;
    default:
        break;
}

spriteBatch.End();
base.Draw(gameTime);
```

Le jeu est dorénavant fonctionnel, vous n'avez plus qu'à mettre votre login dans trouver le bon ordre des lettres pour valider l'épreuve ;)

V - Conclusion

Ce petit tutoriel vous a permis de développer une première application simple de jeu avec XNA sur Windows Phone 7.

Copyright 2010 - Rubrique dotnet. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Droits de diffusion permanents accordés à developpez LLC.