5

Cas pratique : programmer un Pong

Vous disposez à présent des compétences nécessaires pour vous lancer dans la création d'un premier vrai petit jeu. Dans ce chapitre nous allons donc récrire le jeu mythique qu'est devenu Pong. Passage obligé du développeur amateur, ce projet a pour but de vous faire découvrir les pratiques en amont du développement d'un jeu vidéo, puis mettra directement en application toutes les notions découvertes précédemment.

Avant de se lancer dans l'écriture du code

Avant toute chose, et ceci est valable même pour le jeu le plus basique qui soit, vous devez tout planifier. La plupart des projets, dans le monde du jeu vidéo ou ailleurs, échouent parce que cette phase de préparation a été négligée : certains arrivent à terme, mais fournissent un résultat différent de celui attendu, d'autres n'ont même pas cette chance. Même si l'exemple de ce chapitre a l'air simpliste, il vous permettra d'apprendre à prendre les choses en mains dès le début et de gagner du temps pour l'étape de développement.

Définir le principe du jeu

Pong est un jeu inspiré du tennis de table qui a fait son apparition dans les années 1970, d'abord sur une borne d'arcade puis en console de salon.

Il existe certainement des milliers de versions de Pong. Comme nous l'avons précisé dans l'introduction de ce chapitre, il s'agit du premier jeu que la plupart des développeurs réalisent. L'engouement que les programmeurs ont pour ce jeu contribue sans cesse à son

renouvellement : on peut même trouver des versions massivement multi-joueur de Pong sur l'Internet.

Le principe que nous allons implémenter est simple : il y a deux joueurs, chacun contrôlant une raquette. Les deux joueurs s'envoient une balle qui rebondit sur les bords haut et bas de l'écran. Si la balle touche la raquette d'un des deux joueurs, elle repart vers l'autre joueur, cependant, si elle manque la raquette, l'autre joueur gagne.

La version originale de Pong était plus complète puisqu'elle possédait son et affichage du score : deux facettes d'XNA que nous n'avons pas encore abordées.

Formaliser en pseudo-code

Maintenant que le principe de jeu est clair, il est possible de traduire le déroulement d'une partie en pseudo-code. Ce cycle se répétant tant que le joueur n'aura pas quitté le jeu.

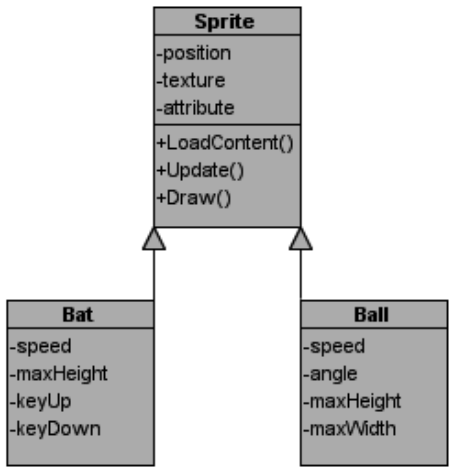
```
Tant que le joueur n'a pas lancé la partie
    Attendre
Fin tant que

Tant que la balle ne sort pas par la gauche ou la droite de l'écran
    Si la balle touche le haut ou le bas de l'écran
        Faire rebondir la balle
    Fin si
    Si la balle touche une raquette
        Faire rebondir la balle
    Fin si
Fin Tant que
```

La figure 5-1 représente les classes `bat` (raquette) et `ball` (balle) ainsi que leurs différents champs. Ces deux classes héritent de la classe `Sprite`.

Figure 5-1

Diagramme des classes bat et ball



L'arrière-plan du jeu pourrait être géré en créant un simple sprite, puisqu'il n'a pas de propriétés particulières. Cependant, pour factoriser le code de notre classe `Game`, nous implémenterons tout de même une classe qui lui sera dédiée. Cette classe dérivera de la classe `DrawableGameComponent`, qui hérite elle-même de la classe `GameComponent` et qui en plus implémente l'interface `IDrawable`, nous fournissant la méthode `Draw()`.

Développement du jeu

Maintenant que vous avez précisé vos objectifs, il est temps de passer à la pratique et de les réaliser.

Création du projet

Dans ce chapitre, il sera question d'un projet pour Windows, mais vous pourrez facilement l'adapter pour Xbox 360 en vous aidant du chapitre précédent si c'est nécessaire.

1. Commencez par créer un nouveau projet que vous baptiserez « Pong ».
2. Renommez le fichier `Game1.cs` en `Pong.cs` et, lorsque Visual Studio vous demande si vous souhaitez également renommer toutes les références à « Game1 », acceptez. Votre classe `Game1` s'appelle maintenant `Pong`.

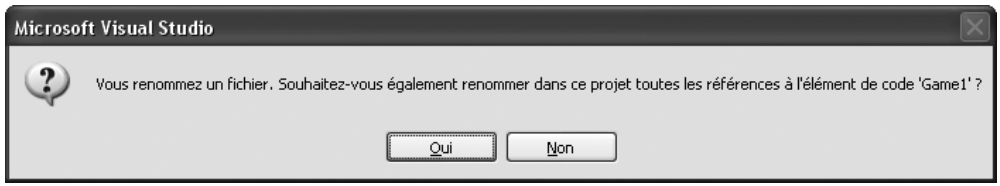


Figure 5-2

Visual Studio peut renommer automatiquement toutes les références à un élément

3. Ensuite, ajoutez les éléments développés dans les chapitres précédents, c'est-à-dire les fichiers `IKeyboardService.cs`, `KeyboardService.cs`, `ServiceHelper.cs` et `Sprite.cs`.
4. Dans votre classe `Pong`, pensez à définir la propriété `Game` de la classe `ServiceHelper`, et à ajouter le composant `KeyboardService` à la collection.

```
public Pong()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
    ServiceHelper.Game = this;
    Components.Add(new KeyboardService(this));
}
```

Votre projet doit donc à présent ressembler à celui visible sur la figure 5-3.

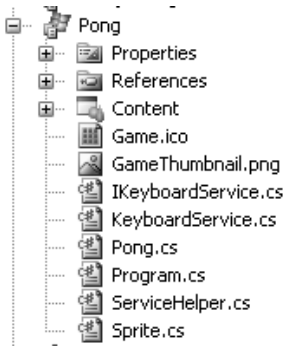


Figure 5-3

Vous récupérerez souvent des éléments développés précédemment pour vos nouveaux projets

L'arrière-plan

Ajoutez une nouvelle classe au projet que vous nommerez `Background` et qui dérivera de `DrawableGameComponent`.

```
class Background : DrawableGameComponent
{
    public Background(Game game)
        : base(game)
    {
    }

    public override void Initialize()
    {
        base.Initialize();
    }

    protected override void LoadContent()
    {
        base.LoadContent();
    }

    public override void Draw(GameTime gameTime)
    {
        base.Draw(gameTime);
    }
}
```

Pour pouvoir dessiner l'arrière-plan, vous aurez besoin d'un `SpriteBatch` puisque vous ne disposez pas de celui présent dans la classe `Pong`. Ajoutez donc un champ de ce type que vous initialiserez dans la méthode `Initialize()`. Vous pouvez récupérer l'objet `GraphicsDevice` de la classe `Pong` en utilisant la propriété `Game` que met à disposition la classe parente.

```
SpriteBatch spriteBatch;

public override void Initialize()
{
spriteBatch = new SpriteBatch(Game.GraphicsDevice);
base.Initialize();
}
```

Il vous faudra finalement un sprite qui correspondra à l'image de l'arrière-plan. Déclarez un champ de type `Sprite`, chargez l'image voulue pour l'arrière-plan et dessinez-le. Notez que pour le chargement de l'image, vous pouvez récupérer le `ContentManager` de la classe `Pong` via la propriété `Game`.

Voici donc le code final de la classe `Background`.

```
class Background : DrawableGameComponent
{
    Sprite sprite;
    SpriteBatch spriteBatch;

    public Background(Game game)
        : base(game)
    {
    }

    public override void Initialize()
    {
        sprite = new Sprite(new Vector2(0, 0));
        spriteBatch = new SpriteBatch(Game.GraphicsDevice);
        base.Initialize();
    }

    protected override void LoadContent()
    {
        sprite.LoadContent(Game.Content, "back");
        base.LoadContent();
    }

    public override void Draw(GameTime gameTime)
    {
        spriteBatch.Begin();
        sprite.Draw(spriteBatch);
        spriteBatch.End();
        base.Draw(gameTime);
    }
}
```

Puis, dans le constructeur de la classe `Pong`, ajoutez un nouvel objet de type `Background` à la collection `Components`.

```
public Pong()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";

    ServiceHelper.Game = this;
    Components.Add(new KeyboardService(this));
    Components.Add(new Background(this));
}
```

Les raquettes

Il est temps d'écrire le code de la classe `Bat`. Celle-ci dérivera bien sûr de la classe `Sprite`. Implémentez dès maintenant les champs présentés dans le diagramme de classes de la première partie de ce chapitre.

```
class Bat : Sprite
{
    float speed;
    int maxHeight;
    Keys keyUp;
    Keys keyDown;

    public Bat(Vector2 position, int playerNumber, int maxHeight, Keys keyUp, Keys
    ➥keyDown)
        : base(position)
    {
        this.playerNumber = playerNumber;
        this.maxHeight = maxHeight;
        this.keyUp = keyUp;
        this.keyDown = keyDown;
        speed = 0.7f;
    }
}
```

Maintenant, vous devez ajouter une méthode `Update()` à cette classe. Passez-lui un objet `GameTime` en paramètre de manière à moduler la vitesse de déplacement des raquettes. Utilisez donc la classe `ServiceHelper` pour savoir si le joueur a appuyé sur une touche, si c'est le cas, vérifiez que la raquette n'est pas sur l'extrémité haute ou basse de l'écran avant de déplacer le sprite.

Pour vérifier si la raquette est en haut ou en bas de l'écran, rappelez-vous que l'origine du repère à l'écran est située en haut à gauche (figure 5-4) et que la position d'un sprite correspond au coin supérieur gauche de l'image. Dans les calculs, il faut donc considérer que l'extrémité haute de l'écran se situe aux points de coordonnées `(x ; 0)` et l'extrémité basse aux points de coordonnées `(y ; maxHeight - textureHeight)`.

Voici donc le code source de la classe `Bat`.

```
class Bat : Sprite
{
    float speed;
    int maxHeight;
    Keys keyUp;
    Keys keyDown;

    public Bat(Vector2 position, int playerNumber, int maxHeight, Keys keyUp, Keys
    ➥keyDown)
        : base(position)
    {
        this.maxHeight = maxHeight;
        this.keyUp = keyUp;
        this.keyDown = keyDown;
        speed = 0.7f;
    }

    public void Update(GameTime gameTime)
    {
        if (ServiceHelper.Get<IKeyboardService>().IsKeyDown(keyDown))
            if (Position.Y < (maxHeight - Texture.Height))
                Position = new Vector2(Position.X, Position.Y + speed *
                ➥gameTime.ElapsedGameTime.Milliseconds);

        if (ServiceHelper.Get<IKeyboardService>().IsKeyDown(keyUp))
            if (Position.Y > 0)
                Position = new Vector2(Position.X, Position.Y - speed *
                ➥gameTime.ElapsedGameTime.Milliseconds);
    }
}
```

Retrouvez ci-dessous le code source de la classe `Pong` qui utilise deux raquettes. La position des sprites n'est déterminée qu'après avoir chargé leur texture, ce qui est normal puisque cette position dépend de la taille de la texture. Notez enfin que le dessin des raquettes se fait après l'appel à la méthode `Draw()` de la classe parente, uniquement pour respecter l'ordre des éléments à l'écran : l'arrière-plan doit être dessiné avec les autres éléments.

```
public class Pong : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Bat playerOne;
    Bat playerTwo;

    public Pong()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";

        ServiceHelper.Game = this;
        Components.Add(new KeyboardService(this));
```

```
        Components.Add(new Background(this));

        playerOne = new Bat(new Vector2(0, 0), graphics.PreferredBackBufferHeight,
        ➥Keys.A, Keys.Q);
        playerTwo = new Bat(new Vector2(0, 0), graphics.PreferredBackBufferHeight,
        ➥Keys.Up, Keys.Down);
    }

    protected override void Initialize()
    {
        base.Initialize();
    }

    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        playerOne.LoadContent(Content, "bat");
        playerOne.Position = new Vector2(20, graphics.PreferredBackBufferHeight /
        ➥2 - playerOne.Texture.Height / 2);
        playerTwo.LoadContent(Content, "bat");
        playerTwo.Position = new Vector2(770, graphics.PreferredBackBufferHeight /
        ➥2 - playerTwo.Texture.Height / 2);
    }

    protected override void UnloadContent()
    {
    }

    protected override void Update(GameTime gameTime)
    {
        playerOne.Update(gameTime);
        playerTwo.Update(gameTime);
        base.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.CornflowerBlue);

        base.Draw(gameTime);

        spriteBatch.Begin();
        playerOne.Draw(spriteBatch);
        playerTwo.Draw(spriteBatch);
        spriteBatch.End();
    }
}
```

Vous devez également modifier la classe `Sprite` et créer une propriété en lecture pour l'objet texture, sans quoi vous ne pourrez pas récupérer ses dimensions.

```
public Texture2D Texture
{
    get { return texture; }
}
```

La balle

La classe `Ball` dérive également de la classe `Sprite`. Cette fois encore, implémentez les champs présentés dans le diagramme du début de ce chapitre.

```
class Ball : Sprite
{
    float speed;
    Vector2 angle;
    int maxHeight;
    int maxWidth;

    public Ball(Vector2 position, int maxWidth, int maxHeight)
        : base(position)
    {
        this.maxHeight = maxHeight;
        this.maxWidth = maxWidth;
        speed = 0.2f;
        angle = new Vector2(1, 1);
    }
}
```

Écrivons maintenant la méthode `Update()`. Celle-ci prendra en paramètre les raquettes des deux joueurs, ainsi qu'une référence vers un booléen qui détermine si le jeu est en pause ou non.

En premier lieu, calculez la nouvelle position de la balle en fonction de la vitesse, du temps et de la direction.

```
Position = new Vector2(Position.X + speed * angle.X *
➥gameTime.ElapsedGameTime.Milliseconds, Position.Y + speed * angle.Y *
➥gameTime.ElapsedGameTime.Milliseconds);
```

Récupérer les raquettes des joueurs vous sera utile pour construire des objets `Rectangle`. Ces objets disposent d'une méthode `Intersects()` vous permettant de détecter les collisions entre la balle et les raquettes. Un objet `Rectangle` se construit tout simplement à partir de deux coordonnées `x` et `y`, une largeur et une hauteur.

```
Rectangle ballRect = new Rectangle((int)Position.X, (int)Position.Y, Texture.Width,
➥Texture.Height);
Rectangle playerOneRect = new Rectangle((int)playerOne.Position.X,
➥(int)playerOne.Position.Y, playerOne.Texture.Width, playerOne.Texture.Height);
Rectangle playerTwoRect = new Rectangle((int)playerTwo.Position.X,
➥(int)playerTwo.Position.Y, playerOne.Texture.Width, playerOne.Texture.Height);
```

Il reste maintenant à tester les différents cas de figure : si la balle touche le haut de la fenêtre ou le bas, si elle entre en collision avec une raquette ou bien si elle a atteint les extrémités gauche ou droite de l'écran. Dans le cas d'une collision, on modifiera la direction.

```
class Ball : Sprite
{
    float speed;
    Vector2 angle;
    int maxHeight;
    int maxWidth;

    public Ball(Vector2 position, int maxWidth, int maxHeight)
        : base(position)
    {
        this.maxHeight = maxHeight;
        this.maxWidth = maxWidth;
        speed = 0.2f;
        angle = new Vector2(1, 1);
    }

    public void Update(GameTime gameTime, Sprite playerOne, Sprite playerTwo, ref
    ➥bool started)
    {
        Position = new Vector2(Position.X + speed * angle.X *
        ➥gameTime.ElapsedGameTime.Milliseconds, Position.Y + speed * angle.Y *
        ➥gameTime.ElapsedGameTime.Milliseconds);

        Rectangle ballRect = new Rectangle((int)Position.X, (int)Position.Y,
        ➥Texture.Width, Texture.Height);
        Rectangle playerOneRect = new Rectangle((int)playerOne.Position.X, (int)
        ➥playerOne.Position.Y, playerOne.Texture.Width, playerOne.Texture.Height);
        Rectangle playerTwoRect = new Rectangle((int)playerTwo.Position.X, (int)
        ➥playerTwo.Position.Y, playerOne.Texture.Width, playerOne.Texture.Height);

        // est-ce qu'il y a collision avec le haut de l'écran ?
        if (Position.Y <= 0)
            angle = new Vector2(angle.X, 1);
        // ... ou bien avec le bas de l'écran ?
        else if (Position.Y >= maxHeight - Texture.Height)
            angle = new Vector2(angle.X, -1);
        // ... ou alors avec le joueur 1 ?
        else if (ballRect.Intersects(playerOneRect))
            angle = new Vector2(1, angle.Y);
        // ... ou bien le joueur 2 ?
        else if (ballRect.Intersects(playerTwoRect))
            angle = new Vector2(-1, angle.Y);
        // ... ou bien l'extrémité gauche ou l'extrémité droite de l'écran a été
        ➥atteinte
        else if (Position.X <= 0 || Position.X + Texture.Width >= maxWidth)
            started = false;
    }
}
```

De retour dans la classe `Pong`, déclarez un objet de type `Ball`, chargez sa texture et dessinez-la. Ajoutez aussi un booléen et initialisez-le à `false`. La méthode `Update()` va légèrement se compliquer. Le programme devra attendre que le joueur appuie sur la barre d'espace avant de lancer la balle et rendre le mouvement des raquettes possible. Dans l'appel à la méthode `Update()` de la balle, vous devez passer une référence vers votre booléen. Ceci se fait grâce à l'instruction `ref`.

```
public class Pong : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Bat playerOne;
    Bat playerTwo;
    Ball ball;
    bool started;

    public Pong()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";

        ServiceHelper.Game = this;
        Components.Add(new KeyboardService(this));
        Components.Add(new Background(this));

        playerOne = new Bat(new Vector2(0, 0), graphics.PreferredBackBufferHeight,
        ➥Keys.A, Keys.Q);
        playerTwo = new Bat(new Vector2(0, 0), graphics.PreferredBackBufferHeight,
        ➥Keys.Up, Keys.Down);

        ball = new Ball(new Vector2(0, 0), graphics.PreferredBackBufferWidth,
        ➥graphics.PreferredBackBufferHeight);

        started = false;
    }

    protected override void Initialize()
    {
        Window.Title = "Appuyez sur espace pour démarrer";

        base.Initialize();
    }

    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        playerOne.LoadContent(Content, "bat");
        playerOne.Position = new Vector2(20, graphics.PreferredBackBufferHeight /
        ➥2 - playerOne.Texture.Height / 2);
        playerTwo.LoadContent(Content, "bat");
```

```
            playerTwo.Position = new Vector2(770, graphics.PreferredBackBufferHeight /
            ➥2 - playerTwo.Texture.Height / 2);
            ball.LoadContent(Content, "ball");
            ball.Position = new Vector2((graphics.PreferredBackBufferWidth / 2) -
            ➥(ball.Texture.Width / 2), (graphics.PreferredBackBufferHeight / 2) -
            ➥(ball.Texture.Height / 2));
        }

        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
        {
            base.Update(gameTime);

            if (started)
            {
                playerOne.Update(gameTime);
                playerTwo.Update(gameTime);
                ball.Update(gameTime, playerOne, playerTwo, ref started);
            }
            else
            {
                if (ServiceHelper.Get<IKeyboardService>().IsKeyDown(Keys.Space))
                {
                    started = true;
                    Window.Title = "Pong !";
                }
            }
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            base.Draw(gameTime);

            spriteBatch.Begin();
            playerOne.Draw(spriteBatch);
            playerTwo.Draw(spriteBatch);
            ball.Draw(spriteBatch);
            spriteBatch.End();
        }
    }
```

Votre jeu est maintenant prêt. Vous pouvez le compiler et l'essayer pendant des heures… Ou peut-être un peu moins !

Figure 5-4
Repère 2D utilisé pour l'affichage

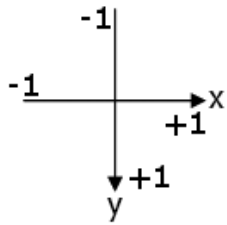
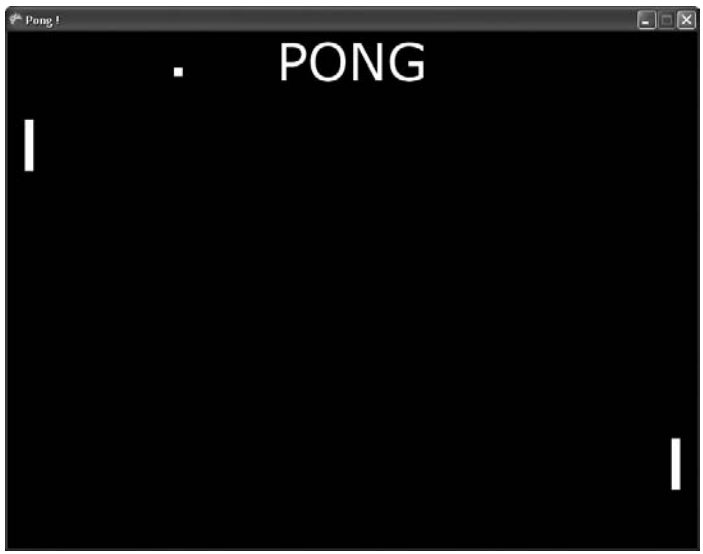


Figure 5-5
Votre premier jeu n'est pas si mal, n'est ce pas ?



Améliorer l'intérêt du jeu

Votre premier jeu est maintenant terminé, mais pour l'instant il n'a absolument rien d'original et est commun à tous les jeux Pong. Pourquoi ne pas essayer d'en augmenter la difficulté ?

Pour commencer, augmentez la vitesse de la balle à chaque fois qu'elle entre en collision avec une raquette.

```
public void Update(GameTime gameTime, Sprite playerOne, Sprite playerTwo, ref bool
➥started)
{
    Position = new Vector2(Position.X + speed * angle.X * gameTime
    ➥.ElapsedGameTime.Milliseconds, Position.Y + speed * angle.Y *
    ➥gameTime.ElapsedGameTime.Milliseconds);

    Rectangle ballRect = new Rectangle((int)Position.X, (int)Position.Y,
    ➥Texture.Width, Texture.Height);
    Rectangle playerOneRect = new Rectangle((int)playerOne.Position.X,
    ➥(int)playerOne.Position.Y, playerOne.Texture.Width, playerOne.Texture.Height);
```

```
        Rectangle playerTwoRect = new Rectangle((int)playerTwo.Position.X,
        ➥(int)playerTwo.Position.Y, playerOne.Texture.Width, playerOne.Texture.Height);

        // est-ce qu'il y a collision avec le haut de l'écran ?
        if (Position.Y <= 0)
            angle = new Vector2(angle.X, 1);
        // ... ou bien avec le bas de l'écran ?
        else if (Position.Y >= maxHeight - Texture.Height)
            angle = new Vector2(angle.X, -1);
        // ... ou alors avec le joueur 1 ?
        else if (ballRect.Intersects(playerOneRect))
        {
            angle = new Vector2(1, angle.Y);
            speed += 0.02f;
        }
        // ... ou bien le joueur 2 ?
        else if (ballRect.Intersects(playerTwoRect))
        {
            angle = new Vector2(-1, angle.Y);
            speed += 0.02f;
        }
        // ... ou alors, l'extrémité gauche ou l'extrémité droite de l'écran a été
        ➥atteinte
        else if (Position.X <= 0 || Position.X + Texture.Width >= maxWidth)
            started = false;
    }
```

Les possibilités d'évolution sont vraiment nombreuses (création d'une intelligence artificielle, ajout de nouvelles balles, etc.), contentez-vous de laisser libre cours à votre imagination, mais gardez toujours à l'esprit que ce n'est pas la débauche de technique qui fait qu'un jeu est bon ou non, le gameplay est vraiment important !

En résumé

Dans ce chapitre, vous avez découvert que la phase de préparation (ou d'élaboration du cahier des charges) ne doit pas être oubliée. Vous avez ensuite mené à bien votre premier projet avec XNA, découvrant au passage comment créer un DrawableComponent et gérer les collisions de façon primitive.