11

Le mode multijoueur

Pong ou *Tennis for Two*, les premiers jeux vidéo, étaient déjà des jeux multijoueurs. Ainsi, le mode multijoueur, qu'il s'agisse de jouer en coopération ou de s'affronter sur un seul et même écran, existe depuis le début des jeux vidéo, et est bien entendu toujours utilisé de nos jours.

Avec le temps, les jeux multijoueurs ont évolué et se sont mis à utiliser les communications réseau, reliant des stations de jeux plus ou moins éloignées (sur le même réseau local ou via Internet).

Dans ce chapitre, nous allons développer un jeu qui utilise le *scrolling* et qui sera d'abord uniquement jouable en solo, puis jouable à deux sur le même écran, et enfin à deux via le réseau.

Jouer à plusieurs sur le même écran

Il existe deux types de jeux vidéo qui offrent une expérience multijoueur sur le même écran : ceux où les joueurs apparaissent les uns à côté des autres dans la même vue et ceux où l'écran est partagé en plusieurs parties.

Dans cette première partie, nous allons voir comment créer un jeu multijoueur utilisant le principe du partage d'écran.

Un jeu en écran partagé, aussi appelé jeu en écran « splitté » (de l'anglais *to split*, qui signifie séparer), est un jeu où l'écran est divisé en plusieurs zones, généralement de même taille, chacune étant réservée à un joueur.

Avec XNA, la division de l'écran en plusieurs parties se fera grâce à des objets de type `Viewport`. Il est possible de fixer la taille de ces objets, ainsi que leur point d'origine, chacun d'entre eux disposera de son propre repère cartésien.

Du mode solo au multijoueur : la gestion des caméras

Au chapitre 3, nous avons vu ce que sont les jeux dits *tile-based* et nous les avons employés pour l'application de l'algorithme A* au chapitre 9. Dans ce chapitre, nous irons encore plus loin avec ce type de jeu et nous allons créer un système de caméra, le but étant de ne pas afficher toute la carte à l'écran, mais d'obtenir un effet de *scrolling* (en français, défilement). Nous avons déjà abordé le *scrolling* au chapitre 6 : il s'agit de faire défiler l'écran sur une carte en fonction des déplacements du joueur.

Créer un jeu solo avec effet de scrolling

Commencez par créer un nouveau projet et récupérez, des précédents projets, les fichiers `IKeyboardService.cs`, `KeyboardService.cs`, `ServiceHelper.cs`, `Map.cs`, `Sprite.cs` et `Tile.cs`.

Tout d'abord, modifiez l'interface `IKeyboardService` et ajoutez-lui la signature d'une méthode `KeyHasBeenPressed`. Le but de cette méthode est de détecter si le joueur a pressé une touche, puis l'a relâchée. Ainsi, le personnage ne se déplacera d'une case qu'à chaque fois que le joueur pressera une touche. Modifiez ensuite la classe `KeyboardService` de manière à ce qu'elle prenne en compte cette nouvelle méthode.

```
interface IKeyboardService
{
    bool IsKeyDown(Keys key);
    bool KeyHasBeenPressed(Keys key);
}
```

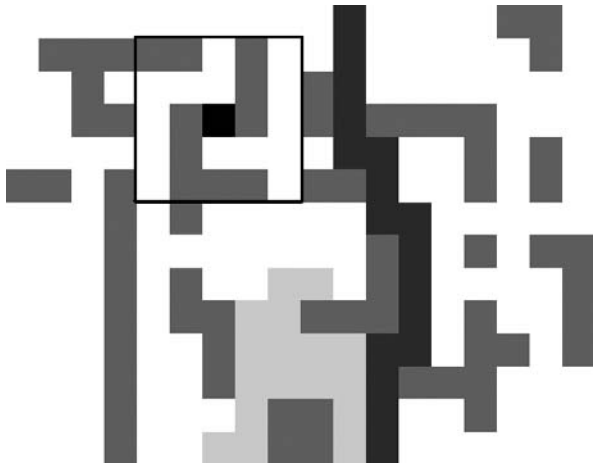
Il n'y a aucune difficulté ici, d'autant plus que vous avez déjà fait la même chose dans un précédent chapitre pour le bouton gauche de la souris.

```
class KeyboardService : GameComponent, IKeyboardService
{
    KeyboardState lastKBState;
    KeyboardState KBState;

    public KeyboardService(Game game)
        : base(game)
    {
        ServiceHelper.Add<IKeyboardService>(this);
    }

    public bool IsKeyDown(Keys key)
    {
        return KBState.IsKeyDown(key);
    }
```

```
    public bool KeyHasBeenPressed(Keys key)
    {
        return lastKBState.IsKeyDown(key) && KBState.IsKeyUp(key);
    }

    public override void Update(GameTime gameTime)
    {
        lastKBState = KBState;
        KBState = Keyboard.GetState();
        base.Update(gameTime);
    }
}
```

Intéressez-vous maintenant au système de caméra. La figure 11-1 représente votre objectif. Vous avez une carte (celle représentée ici est tirée du chapitre 9), et le carré aux bords noirs autour du joueur représente son champ de vision, c'est-à-dire la caméra.

Figure 11-1
Une vision partielle d'une carte grâce à une caméra



Sur cet exemple, le joueur voit l'environnement qui l'entoure sur une distance de deux cases, la caméra est donc un carré de 5×5 cases. La position de la caméra sur la carte sera déterminée par un couple de coordonnées (x, y). Vous pouvez donc utiliser un objet de type `Rectangle` qui dispose de toutes ces caractéristiques.

Le sprite représentant le joueur doit toujours être placé au milieu de l'écran. Lorsque le joueur appuie sur les touches de déplacement, ce n'est pas la position à l'écran du sprite qui est modifiée, mais sa position sur la carte et les coordonnées de l'origine de la caméra.

Modifiez maintenant la méthode `Draw()` de la classe `Map` afin qu'elle ne dessine que les cases qui sont dans le champ de vision de la caméra. Parcourez le tableau `tileList` en bouclant sur les dimensions de l'objet `Rectangle`, puis positionnez les sprites à l'écran en fonction de leur position dans le rectangle de la caméra et appelez leur méthode `Draw()`.

La classe Map modifiée

```
class Map
{
    Tile[,] tileList;

    public Tile[,] TileList
    {
        get { return tileList; }
        set { tileList = value; }
    }

    public Map(byte[,] table)
    {
        tileList = new Tile[table.GetLength(0),table.GetLength(1)];

        for (int y = 0; y < table.GetLength(0); y++)
        {
            for (int x = 0; x < table.GetLength(1); x++)
            {
                tileList[y, x] = new Tile(y, x, table[y, x]);
            }
        }
    }

    public void Draw(SpriteBatch spriteBatch, Rectangle camera)
    {
        for (int y = camera.Y; y < camera.Y + camera.Height; y++)
        {
            for(int x = camera.X; x < camera.X + camera.Width; x++)
            {
                tileList[y, x].Position = new Vector2((x - camera.X) * 32, (y -
                ➥camera.Y) * 32);
                tileList[y, x].Draw(spriteBatch);
            }
        }
    }

    public bool ValidCoordinates(int x, int y)
    {
        if (x < 0)
            return false;

        if (y < 0)
            return false;

        if (x >= tileList.GetLength(1))
            return false;

        if (y >= tileList.GetLength(0))
            return false;

        return true;
    }
}
```

Passez à présent à la classe principale du projet où vous utiliserez ces nouvelles fonctions. Déclarez un objet `Map`, un objet `Tile` qui représentera le héros et enfin un objet `Rectangle` pour la caméra.

Attention avec le tableau de `byte` lorsque vous instanciez l'objet `Map`. En effet, pour éviter qu'une exception soit levée lorsque le joueur est proche des bords de la carte (puisque la méthode `Draw()` de l'objet `Map` essaiera d'accéder à des index qui n'existent pas dans le tableau), placez des murs sur les bords de la carte. N'oubliez pas non plus de placer le sprite représentant le joueur au milieu de l'écran.

Lorsque vous captez une entrée utilisateur, vérifiez si la case vers laquelle il souhaite se déplacer est un mur ou non. Si c'en est un, le déplacement ne doit pas avoir lieu.

Enfin, dans la méthode `Draw()`, pensez à dessiner le joueur après avoir dessiné la carte.

La classe principale du mini-jeu

```
public class Chapitre11 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    Map map;
    Tile herosA;
    Rectangle cameraA;

    public Chapitre11()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        ServiceHelper.Game = this;
        Components.Add(new KeyboardService(this));
        graphics.PreferredBackBufferWidth = 160;
        graphics.PreferredBackBufferHeight = 160;
    }

    protected override void Initialize()
    {
        map = new Map(new byte[,] {
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 1, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 1, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 1, 0, 0, 0, 1, 0},
            {0, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1, 1, 1, 0, 1, 0},
            {0, 1, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 1, 0},
            {0, 1, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 1, 0},
            {0, 1, 3, 0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 2, 2, 0, 0, 1, 0},
            {0, 1, 3, 0, 3, 3, 3, 3, 3, 0, 0, 0, 2, 2, 0, 0, 1, 0},
            {0, 1, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 2, 2, 2, 0, 1, 0},
            {0, 1, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 2, 2, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
            });
```

```
        cameraA = new Rectangle(0, 0, 5, 5);
        herosA = new Tile(2, 2, 4);
        herosA.Position = new Vector2(64, 64);

        base.Initialize();
    }

    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);

        foreach (Tile tile in map.TileList)
        {
            tile.LoadContent(Content, "tile");
        }
        herosA.LoadContent(Content, "tile");
    }

    protected override void Update(GameTime gameTime)
    {
        if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Up))
        {
            if (map.TileList[herosA.Y - 1, herosA.X].Type >= 0)
            {
                cameraA.Y -= 1;
                herosA.Y -= 1;
            }
        }
        if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Right))
        {
            if (map.TileList[herosA.Y, herosA.X + 1].Type >= 0)
            {
                cameraA.X += 1;
                herosA.X += 1;
            }
        }
        if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Down))
        {
            if (map.TileList[herosA.Y + 1, herosA.X].Type >= 0)
            {
                cameraA.Y += 1;
                herosA.Y += 1;
            }
        }
        if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Left))
        {
            if (map.TileList[herosA.Y, herosA.X - 1].Type >= 0)
            {
                cameraA.X -= 1;
                herosA.X -= 1;
            }
        }

        base.Update(gameTime);
    }
```

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

        spriteBatch.Begin();
        map.Draw(spriteBatch, cameraA);
        herosA.Draw(spriteBatch);
        spriteBatch.End();

        base.Draw(gameTime);
    }
}
```

Avant de compiler le projet, ajoutez le sprite qui sera utilisé pour chacune des cases (ici une image blanche de 32 × 32 pixels). Vous pouvez ensuite l'essayer et vous amuser avec le *scrolling*.

Figure 11-2
Votre jeu en mode solo

Adapter les caméras au multijoueur

À présent, nous allons modifier le jeu de manière à ce qu'il puisse accueillir deux joueurs en simultané sur le même écran. Dans le code source de cet ouvrage, le projet correspondant est nommé `Chapitre 11_2`.

La première chose à faire est de vous occuper des objets `Viewport`. Créez-en deux, un pour chaque joueur. Ici, ils seront appelés `viewportA` pour la portion supérieure de l'écran et `viewportB` pour la portion inférieure de l'écran.

Spécifiez les dimensions des deux vues en définissant leurs propriétés `Width` et `Height`. Dans le programme exemple lié à ce chapitre, la largeur d'une vue est la même que la largeur de la fenêtre de jeu, et sa hauteur correspond à la moitié de celle de la fenêtre de jeu. Pensez donc à augmenter la hauteur de la fenêtre de jeu si vous voulez que les deux vues soient des carrés.

Enfin, décalez la deuxième vue en hauteur grâce à sa propriété `Y`. Les deux vues seront donc alignées verticalement ; pour les aligner horizontalement, vous modifierez simplement la propriété `X`.

```
viewportA.Width = graphics.PreferredBackBufferWidth;
viewportA.Height = graphics.PreferredBackBufferHeight / 2;
```

```
viewportB.Y = graphics.PreferredBackBufferHeight / 2;
viewportB.Width = graphics.PreferredBackBufferWidth;
viewportB.Height = graphics.PreferredBackBufferHeight / 2;
```

Déclarez un nouvel objet `Tile` qui représentera le deuxième joueur, ainsi qu'un nouvel objet `Rectangle` pour la caméra du deuxième joueur et faites-les bouger lorsque des touches du clavier seront utilisées (ici les touches Z-Q-S-D).

Xbox 360

Dans l'exemple de jeu qui vous est proposé ici, les deux joueurs utilisent le clavier. Vous pouvez modifier le projet pour qu'il soit utilisable avec les manettes de la Xbox 360, vous différencierez les entrées utilisateur grâce à l'énumération `PlayerIndex`.

Sur Xbox 360, les manettes peuvent être complétées par des claviers. L'état de ces claviers se récupère de la manière suivante :

```
Keyboard.GetState(PlayerIndex.Two);
```

Enfin, vous n'avez plus qu'à dessiner dans les deux vues. Pour vous placer sur une vue, modifiez la propriété `Viewport` de l'objet `GraphicsDevice`. Le dessin des sprites se fait de manière classique.

La classe principale du jeu multijoueur

```
public class Chapitre11_2 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    Map map;
    Tile herosA;
    Tile herosB;

    Rectangle cameraA;
    Rectangle cameraB;

    Viewport viewportA;
    Viewport viewportB;

    public Chapitre11_2()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        ServiceHelper.Game = this;
        Components.Add(new KeyboardService(this));
        graphics.PreferredBackBufferWidth = 160;
        graphics.PreferredBackBufferHeight = 320;
    }

    protected override void Initialize()
    {
```

```
        map = new Map(new byte[,] {
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 1, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 1, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 1, 0, 0, 0, 1, 0},
            {0, 1, 3, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1, 1, 1, 0, 1, 0},
            {0, 1, 3, 3, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 1, 0},
            {0, 1, 3, 3, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 1, 0},
            {0, 1, 3, 0, 0, 0, 3, 3, 0, 0, 0, 0, 2, 2, 0, 0, 1, 0},
            {0, 1, 3, 0, 3, 3, 3, 3, 3, 0, 0, 0, 2, 2, 0, 0, 1, 0},
            {0, 1, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 2, 2, 2, 0, 1, 0},
            {0, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 2, 2, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
            });

        cameraA = new Rectangle(0, 0, 5, 5);
        cameraB = new Rectangle(1, 1, 5, 5);

        herosA = new Tile(cameraA.X + 2, cameraA.Y + 2, 4);
        herosA.Position = new Vector2(64, 64);

        herosB = new Tile(cameraB.X + 2, cameraB.Y + 2, 4);
        herosB.Position = new Vector2(64, 64);

        viewportA.Width = graphics.PreferredBackBufferWidth;
        viewportA.Height = graphics.PreferredBackBufferHeight / 2;

        viewportB.Y = graphics.PreferredBackBufferHeight / 2;
        viewportB.Width = graphics.PreferredBackBufferWidth;
        viewportB.Height = graphics.PreferredBackBufferHeight / 2;

        base.Initialize();
    }

    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);

        foreach (Tile tile in map.TileList)
        {
            tile.LoadContent(Content, "tile");
        }
        herosA.LoadContent(Content, "tile");
        herosB.LoadContent(Content, "tile");
    }

    protected override void Update(GameTime gameTime)
    {
        if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Up))
        {
            if (map.TileList[herosA.Y - 1, herosA.X].Type >= 0)
            {
                cameraA.Y -= 1;
```

```
            herosA.Y -= 1;
        }
    }
    if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Right))
    {
        if (map.TileList[herosA.Y, herosA.X + 1].Type >= 0)
        {
            cameraA.X += 1;
            herosA.X += 1;
        }
    }
    if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Down))
    {
        if (map.TileList[herosA.Y + 1, herosA.X].Type >= 0)
        {
            cameraA.Y += 1;
            herosA.Y += 1;
        }
    }
    if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Left))
    {
        if (map.TileList[herosA.Y, herosA.X - 1].Type >= 0)
        {
            cameraA.X -= 1;
            herosA.X -= 1;
        }
    }
    if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Z))
    {
        if (map.TileList[herosB.Y - 1, herosB.X].Type >= 0)
        {
            cameraB.Y -= 1;
            herosB.Y -= 1;
        }
    }
    if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.D))
    {
        if (map.TileList[herosB.Y, herosB.X + 1].Type >= 0)
        {
            cameraB.X += 1;
            herosB.X += 1;
        }
    }
    if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.S))
    {
        if (map.TileList[herosB.Y + 1, herosB.X].Type >= 0)
        {
            cameraB.Y += 1;
            herosB.Y += 1;
        }
    }
    if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.Q))
    {
```

```
            if (map.TileList[herosB.Y, herosB.X - 1].Type >= 0)
            {
                cameraB.X -= 1;
                herosB.X -= 1;
            }
        }

        base.Update(gameTime);
    }

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

        GraphicsDevice.Viewport = viewportA;
        spriteBatch.Begin();
        map.Draw(spriteBatch, cameraA);
        herosA.Draw(spriteBatch);
        spriteBatch.End();

        GraphicsDevice.Viewport = viewportB;
        spriteBatch.Begin();
        map.Draw(spriteBatch, cameraB);
        herosB.Draw(spriteBatch);
        spriteBatch.End();

        base.Draw(gameTime);
    }
}
```

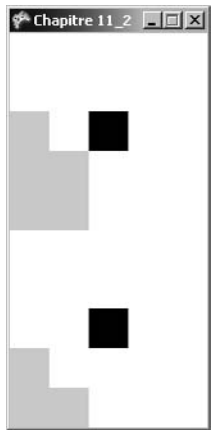
Avancé

Rien ne vous empêche d'utiliser le même objet `SpriteBatch` pour dessiner dans différents `Viewport`. Cependant, vous devrez faire un appel à `Begin()` et à `End()` pour chacun des objets `Viewport` !

Vous pouvez à présent essayer le jeu. Les deux joueurs ont chacun une portion d'écran qui leur est réservée : ils peuvent se déplacer sans problèmes, mais sans se voir (figure 11-3).

Figure 11-3

Les deux joueurs ne se voient pas



Il vous faut donc créer deux nouveaux objets `Tile` qui correspondront aux représentations d'un joueur sur la vue de l'autre et inversement.

Pour commencer, créez un nouveau type de `Tile` pour que les deux joueurs n'apparaissent pas de la même couleur sur la même vue. Modifiez simplement le constructeur de la classe et ajoutez une nouvelle branche au `switch`.

```
public Tile(int y, int x, byte type)
    : base(new Vector2(x * 32, y * 32))
{
    this.x = x;
    this.y = y;

    switch (type)
    {
        case 1:
            Color = Color.Gray;
            this.type = TileType.Wall;
            break;
        case 3:
            Color = Color.LightGreen;
            this.type = TileType.Tree;
            break;
        case 2:
            Color = Color.Blue;
            this.type = TileType.Water;
            break;
        case 4:
            Color = Color.Black;
            this.type = TileType.Human;
            break;
        case 5:
            Color = Color.Red;
            this.type = TileType.Human;
            break;
        default:
            this.type = TileType.Normal;
            break;
    }
}
```

Dans une vue, le personnage que le joueur ne contrôle pas apparaîtra donc en rouge.

Lors du dessin de la vue du joueur A, vérifiez que le joueur B est dans le champ de la caméra A. Si c'est le cas, modifiez la position à l'écran de sa représentation et dessinez-la. Faites le même traitement pour le joueur A sur la vue du joueur B.

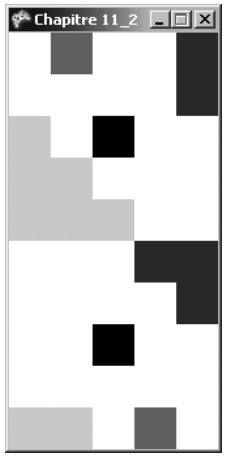
Les ajouts à effectuer dans la classe principale

```
Tile herosBOnA;
Tile herosAOnB;
```

```
protected override void Initialize()
{
    // …
    herosBOnA = new Tile(0, 0, 5);
    herosAOnB = new Tile(0, 0, 5);
}

protected override void LoadContent()
{
    // …
    herosAOnB.LoadContent(Content, "tile");
    herosBOnA.LoadContent(Content, "tile");
}

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

    GraphicsDevice.Viewport = viewportA;
    spriteBatch.Begin();
    map.Draw(spriteBatch, cameraA);
    herosA.Draw(spriteBatch);
    if (cameraA.Contains(herosB.X, herosB.Y))
    {
        herosBOnA.Position = new Vector2((herosB.X - cameraA.X) * 32, (herosB.Y -
        ➥cameraA.Y) * 32);
        herosBOnA.Draw(spriteBatch);
    }
    spriteBatch.End();

    GraphicsDevice.Viewport = viewportB;
    spriteBatch.Begin();
    map.Draw(spriteBatch, cameraB);
    herosB.Draw(spriteBatch);
    if (cameraB.Contains(herosA.X, herosA.Y))
    {
        herosAOnB.Position = new Vector2((herosA.X - cameraB.X) * 32, (herosA.Y -
        ➥cameraB.Y) * 32);
        herosAOnB.Draw(spriteBatch);
    }
    spriteBatch.End();

    base.Draw(gameTime);
}
```

À présent, vous pouvez tester le jeu, il fonctionne à merveille (figure 11-4).

Figure 11-4
Cette fois-ci, les deux joueurs peuvent se croiser sans problèmes



Personnaliser les différentes vues

Maintenant que vous connaissez le B.A-BA des vues avec XNA, voyons quelques précisions sur leur fonctionnement. Nous laisserons l'exemple de jeu précédent de côté.

Pour nettoyer le contenu d'une vue et lui appliquer une couleur, sélectionnez la vue concernée puis, comme vous le faites d'ordinaire pour la totalité de l'écran, appelez la méthode `Clear()` de l'objet `GraphicsDevice` et passez-lui une couleur.

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

    GraphicsDevice.Viewport = viewA;
    GraphicsDevice.Clear(Color.Yellow);

    base.Draw(gameTime);
}
```

Vos vues ne doivent pas forcément remplir toute la surface de l'écran. Vous pouvez les placer où vous le voulez et leur donner des dimensions farfelues si vous le désirez (figure 11-5).

Différentes tailles de vues

```
public class Chapitre11_3 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    Viewport viewA;
    Viewport viewB;
```

Figure 11-5
Les vues se placent où vous le voulez

```
    public Chapitre11_3()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
    }

    protected override void Initialize()
    {
        viewA.Width = 100;
        viewA.Height = 100;
        viewA.X = 50;

        viewB.Width = 300;
        viewB.Height = 100;
        viewB.X = 250;
        viewB.Y = 100;

    base.Initialize();
    }

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

        GraphicsDevice.Viewport = viewA;
        GraphicsDevice.Clear(Color.Yellow);

        GraphicsDevice.Viewport = viewB;
        GraphicsDevice.Clear(Color.Green);

        base.Draw(gameTime);
    }
}
```

Même si vous utilisez une vue, rien ne vous empêche de dessiner d'une manière plus classique sur la totalité de la fenêtre (figure 11-6).

```
public class Chapitre11_3 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    Viewport view;

    Sprite sprite;
    Sprite sprite2;

    public Chapitre11_3()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
    }

    protected override void Initialize()
    {
        view.Width = 300;
        view.Height = 100;
        view.X = 250;
        view.Y = 100;

        sprite = new Sprite(Vector2.Zero);
        sprite2 = new Sprite(Vector2.Zero);

        base.Initialize();
    }

    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        sprite.LoadContent(Content, "GameThumbnail");
        sprite2.LoadContent(Content, "GameThumbnail");
    }

    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.CornflowerBlue);
        spriteBatch.Begin();
        sprite2.Draw(spriteBatch);
        spriteBatch.End();

        GraphicsDevice.Viewport = view;
        GraphicsDevice.Clear(Color.Green);
        spriteBatch.Begin();
        sprite.Draw(spriteBatch);
        spriteBatch.End();

        base.Draw(gameTime);
    }
}
```

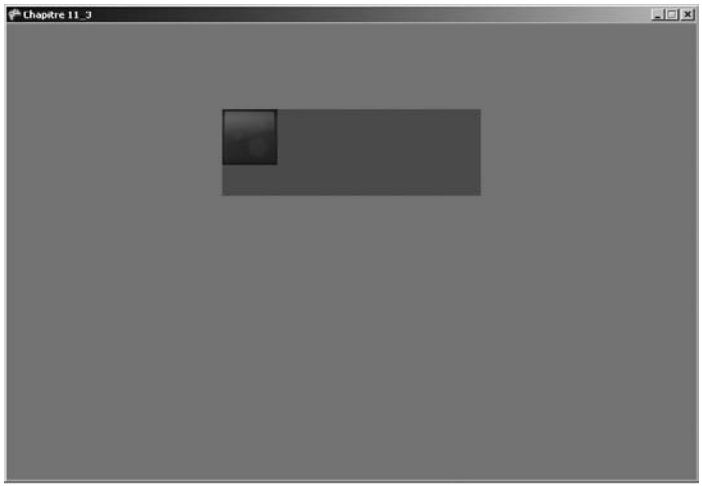
Figure 11-6

Affichage classique dans la fenêtre tout en utilisant une vue

Toutefois, faites attention à l'ordre des dessins. Si vous voulez dessiner dans la fenêtre principale après avoir utilisé une vue particulière, le code suivant ne fonctionnera pas (figure 11-7).

Figure 11-7

De cette manière, vous ne pouvez pas redessiner dans la vue principale




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

    GraphicsDevice.Viewport = view;
    GraphicsDevice.Clear(Color.Green);
    spriteBatch.Begin();
    sprite.Draw(spriteBatch);
```

```
        spriteBatch.End();

        spriteBatch.Begin();
        sprite2.Draw(spriteBatch);
        spriteBatch.End();

        base.Draw(gameTime);
    }
```

Vous devrez enregistrer la vue principale pour pouvoir la réutiliser plus loin.

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    Viewport mainView = GraphicsDevice.Viewport;

    GraphicsDevice.Viewport = view;
    GraphicsDevice.Clear(Color.Green);
    spriteBatch.Begin();
    sprite.Draw(spriteBatch);
    spriteBatch.End();

    GraphicsDevice.Viewport = mainView;
    spriteBatch.Begin();
    sprite2.Draw(spriteBatch);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

Si vous désirez faire jouer plus de quatre joueurs dans la même partie ou si vous désirez connecter des joueurs à travers le réseau, vous ne pourrez pas vous contenter du jeu en écran séparé.

Le multijoueur en réseau

Dans cette deuxième partie, nous allons nous occuper des mécanismes qui vous permettront de réaliser un jeu multijoueur en réseau. Le but est ici de connecter de nombreux joueurs situés à des emplacements géographiques différents. Cependant, vous devrez faire face à de nouvelles contraintes, telles que la gestion des sessions ou encore l'envoi d'informations via le réseau.

S'appuyer sur la plate-forme Live

Windows Live est le nom de la plate-forme de services Internet de Microsoft. Le point fort de ces services est qu'ils utilisent tous un système d'authentification unifié : les comptes Live.

La gestion du réseau avec XNA est basée sur ce système de comptes Live. On distingue deux types de comptes :

- Les comptes Live, enregistrés sur les serveurs de Microsoft et qui sont liés à un *gamer tag*. Ce type de compte est nécessaire pour créer un jeu multijoueur et pour pouvoir y jouer.
- Les comptes locaux, qui, comme leur nom l'indique, ne peuvent être utilisés que localement et qui n'offrent qu'un accès restreint aux services de la plate-forme Live.

Pour utiliser le réseau avec XNA, vous passerez par les *gamer services*. Pour mémoire, ils s'initialisent de la manière suivante :

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

En phase de test

Lorsque vous testerez les programmes qui utilisent le réseau avec XNA, vous devrez utiliser plusieurs machines (PC, Xbox ou Zune). En effet, il est impossible de lancer plusieurs instances d'une application qui utilise les *gamer services* sur la même machine.

Comme nous avons déjà vu au chapitre 8 comment ouvrir les différents menus de ces services, il n'en sera donc pas question ici.

Implémenter les fonctionnalités de jeu en réseau

Une partie en réseau, ou session, se gère via un objet de type `NetworkSession`.

Les sessions et la connexion à la plate-forme de jeu en réseau

La première méthode de cette classe qu'il vous faut utiliser est la méthode `Create()`. C'est elle qui permet de créer une partie en réseau. Le tableau 11-1 répertorie les paramètres les plus courants pour cette méthode :

Tableau 11-1 Paramètres du premier constructeur de Create()

Paramètre	Description
`NetworkSessionType sessionType`	Type de la session à créer (réseau local, interne à la machine, classement sur Internet, etc.).
`Int maxLocalGamers`	Nombre maximal de comptes locaux autorisés dans la partie.
`Int maxGamers`	Nombre de joueurs maximal.

Une fois la session créée, appelez sa méthode `Update()` à chaque frame. Cette méthode réalise les actions suivantes :

- envoyer les paquets réseaux ;
- mettre à jour l'état de la session ;

- récupérer les paquets réseau entrants.

L'appel de cette méthode vous épargne les soucis de threads et de synchronisation que vous auriez pu rencontrer en programmant vous-même la gestion du réseau.

Dans l'exemple suivant, nous allons créer une partie sur le réseau local lorsque le joueur appuiera sur la touche C de son clavier. Notez que vous pouvez récupérer des informations sur la partie grâce aux propriétés de l'objet `NetworkSession`. Dans le code ci-dessous, une vérification de la propriété `IsHost` permet de déterminer si l'instance du programme héberge la partie ou non.

Exemple de création d'une partie sur le réseau local

```
public class Chapitre11_Server : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    NetworkSession session;

    public Chapitre11_Server()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        ServiceHelper.Game = this;
        Components.Add(new KeyboardService(this));
        Components.Add(new GamerServicesComponent(this));
    }

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

    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
    }

    protected override void Update(GameTime gameTime)
    {
        if (ServiceHelper.Get<IKeyboardService>().KeyHasBeenPressed(Keys.C) &&
        ➥session == null)
            session = NetworkSession.Create(NetworkSessionType.SystemLink, 2, 31);

        if (session != null)
        {
            if (session.IsHost)
                Window.Title = "Je suis le serveur (" + session.SessionState
                ➥.ToString() + ")";

            session.Update();
        }

        base.Update(gameTime);
    }
```

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

        base.Draw(gameTime);
    }
}
```

Identifier les parties en cours disponibles

Pour rejoindre une partie, vous devez d'abord rechercher une liste de parties disponibles. Pour cela, vous disposez de la méthode statique `Find()` de la classe `NetworkSession`, ou bien, si vous souhaitez utiliser des processus asynchrones, des méthodes `BeginFind()` et `EndFind()`. Comme nous l'avons vu au chapitre 8, la méthode synchrone bloque le jeu, alors que la méthode asynchrone effectue le traitement en arrière-plan. Là encore, vous devez spécifier le type de partie réseau. Vous pouvez ajouter un paramètre `NetworkSession Properties`, qui filtre la liste de parties disponibles.

Dans les deux cas, vous récupérerez une collection de `AvailableNetworkSession` de type `AvailableNetworkSessionCollection`.

Pour rejoindre une partie, vous n'avez plus qu'à utiliser la méthode `Join` de la classe `NetworkSession`.

Dans l'exemple suivant, si le joueur appuie sur la touche C (ou le bouton A de la manette Xbox), un serveur est créé. S'il appuie sur S (ou B), le programme se recherche les parties disponibles et, si la recherche retourne au moins un résultat, il s'y connecte.

Si le programme est client sur un serveur, il affiche le nom du serveur dans sa barre de titre, sinon il affiche « Je suis le serveur » (figure 11-8).

Création, recherche de parties et connexion

```
public class Chapitre11_4 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    NetworkSession session;
    AvailableNetworkSessionCollection availableSessions;

    public Chapitre11_4()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        Components.Add(new GamerServicesComponent(this));
    }

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

```
    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
    }

    protected override void Update(GameTime gameTime)
    {
        if ((Keyboard.GetState().IsKeyDown(Keys.C) || GamePad.GetState
        ➥(PlayerIndex.One).IsButtonDown(Buttons.A)) && session == null)
            session = NetworkSession.Create(NetworkSessionType.SystemLink, 2, 31);

        if ((Keyboard.GetState().IsKeyDown(Keys.S) || GamePad.GetState
        ➥(PlayerIndex.One).IsButtonDown(Buttons.B)) &&
        ➥SignedInGamer.SignedInGamers.Count != 0 && session == null)
        {
            availableSessions = NetworkSession.Find(NetworkSessionType.SystemLink,
            ➥2, null);

            if (availableSessions != null && availableSessions.Count > 0)
                session = NetworkSession.Join(availableSessions[0]);
        }

        if (session != null)
        {
            if (session.IsHost)
                Window.Title = "Je suis le serveur";
            else
                Window.Title = "Je suis client sur la partie de " +
                ➥session.Host.ToString();

            session.Update();
        }

        base.Update(gameTime);
    }

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

        base.Draw(gameTime);
    }
}
```

Figure 11-8

Cette instance du jeu est correctement connectée à une partie en réseau

Transmettre les données : la partie est en cours

Pour envoyer des données, vous utilisez un objet de type `PacketWriter`. Vous y entrez les données à envoyer (un très grand nombre de types de données est supporté), puis passez cet objet à la méthode `SendData ()` du joueur qui envoie les données. Cette méthode attend un autre paramètre correspondant au mode de transmission des données (voir le tableau 11-2).

> **En pratique**
> Tous les joueurs (y compris l'expéditeur) reçoivent ces données.

Tableau 11-2 Méthodes d'expédition

Nom	Description
`SendDataOptions.Chat`	Les données envoyées correspondent à une conversation entre joueurs.
`SendDataOptions.InOrder`	Les données peuvent être perdues sur le réseau, mais lorsqu'elles arrivent, elles sont toujours dans l'ordre où elles ont été envoyées.
`SendDataOptions.None`	Les données peuvent être perdues et arriver dans le désordre.
`SendDataOptions.Reliable`	Les données ne peuvent pas être perdues, mais leur ordre d'arrivée n'est pas garanti.
`SendDataOptions.ReliableInOrder`	Les données ne peuvent pas être perdues et arriveront dans le bon ordre.

Bien évidemment, si vous utilisez une option qui garantit une bonne intégrité de la transmission (pas de perte et arrivée des données dans le bon ordre), les performances sont altérées.

Pour recevoir des données, recourez à un objet de type `PacketReader`. Lorsque des données viennent d'arriver, la propriété `IsDataAvailable` du joueur local (objet de type `LocalNetworkGamer`) est à `true`.

Vous utiliserez ensuite la méthode `ReceiveData ()` du joueur local pour récupérer ces données. Vous devez passer à cette méthode une référence à un objet de type `NetworkGamer`, qui correspond à l'expéditeur du paquet.

Ensuite, pour extraire les données du `PacketReader`, pensez à la méthode `ReadType ()`, où `Type` est le type des données à extraire.

La méthode `Dispose()` de l'objet `NetworkSession` sert à quitter une partie multijoueur. Ensuite, mettez sa référence (et éventuellement celle de la collection `AvailableNetworkSessionCollection`) à `null`.

L'exemple de code ci-dessous est une amélioration de l'exemple précédent : il vous permet, lorsque vous appuyez sur la touche M du clavier (ou le bouton Y de la manette), d'envoyer l'heure courante sous forme de chaîne de caractères à tous les joueurs (sauf

l'expéditeur) connectés au serveur. Lorsque vous appuyez sur Q (ou le bouton Back dans le cas de la manette), il ferme la connexion à la partie.

Exemple de dialogue entre client et serveur

```
public class Chapitre11_4 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    SpriteFont font;

    NetworkSession session;
    AvailableNetworkSessionCollection availableSessions;

    PacketReader packetReader;
    PacketWriter packetWriter;

    string lastStringReceived = "";

    public Chapitre11_4()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        Components.Add(new GamerServicesComponent(this));
    }

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

        packetReader = new PacketReader();
        packetWriter = new PacketWriter();
    }

    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        font = Content.Load<SpriteFont>("font");
    }

    protected override void Update(GameTime gameTime)
    {
        if ((Keyboard.GetState().IsKeyDown(Keys.Q) || GamePad.GetState
        ➥(PlayerIndex.One).IsButtonDown(Buttons.Back)) && session != null)
        {
            session.Dispose();
            session = null;
            availableSessions = null;
        }

        if ((Keyboard.GetState().IsKeyDown(Keys.C) || GamePad.GetState
        ➥(PlayerIndex.One).IsButtonDown(Buttons.A)) && session == null)
            session = NetworkSession.Create(NetworkSessionType.SystemLink, 2, 31);
```

```
        if ((Keyboard.GetState().IsKeyDown(Keys.S) || GamePad.GetState
        ➥(PlayerIndex.One).IsButtonDown(Buttons.B)) &&
        ➥SignedInGamer.SignedInGamers.Count != 0 && session == null)
        {
            availableSessions = NetworkSession.Find(NetworkSessionType.SystemLink,
            ➥2, null);

            if (availableSessions != null && availableSessions.Count > 0)
                session = NetworkSession.Join(availableSessions[0]);
        }

        if ((Keyboard.GetState().IsKeyDown(Keys.M) || GamePad.GetState
        ➥(PlayerIndex.One).IsButtonDown(Buttons.Y)) && session != null)
        {
            packetWriter.Write(DateTime.Now.ToString());
            session.LocalGamers[0].SendData(packetWriter, SendDataOptions.);
        }

        if (session != null)
        {
            session.Update();

            LocalNetworkGamer gamer = session.LocalGamers[0];
            if (gamer.IsDataAvailable)
            {
                NetworkGamer sender;
                gamer.ReceiveData(packetReader, out sender);
                if (gamer != sender)
                    lastStringReceived = packetReader.ReadString();
            }
        }

        base.Update(gameTime);
    }

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

        spriteBatch.Begin();

        if (session != null)
        {
            if (session.IsHost)
                spriteBatch.DrawString(font, "Serveur (" + SignedInGamer
                ➥.SignedInGamers[0].Gamertag + ")", new Vector2(100, 100),
                ➥Color.White);
            else
                spriteBatch.DrawString(font, "Client (" + SignedInGamer
                ➥.SignedInGamers[0].Gamertag + ") connecte sur " +
                ➥session.Host.ToString() , new Vector2(100, 100), Color.White);
```

```
            spriteBatch.DrawString(font, session.AllGamers.Count + " joueur(s)
            ➥connecte(s)", new Vector2(100, 150), Color.White);

            for (int i = 0; i < session.AllGamers.Count; i++)
            {
                spriteBatch.DrawString(font, session.AllGamers[i].Gamertag, new
                ➥Vector2(100, 200 + (i * 50)), Color.White);
            }

            spriteBatch.DrawString(font, "Dernier message recu", new Vector2(500,
            ➥100), Color.White);
            spriteBatch.DrawString(font, lastStringReceived, new Vector2(500, 150),
            ➥Color.White);

            session.Update();
        }

        spriteBatch.End();

        base.Draw(gameTime);
    }
}
```

La figure 11-9 présente l'écran du serveur (sur la Xbox 360) et sur la figure 11-10, l'écran du client (sur l'ordinateur). Dans les deux cas, vous retrouvez la liste des joueurs présents dans la session, ainsi que la date du dernier message reçu.

Figure 11-9

L'écran côté serveur (sur Xbox 360)

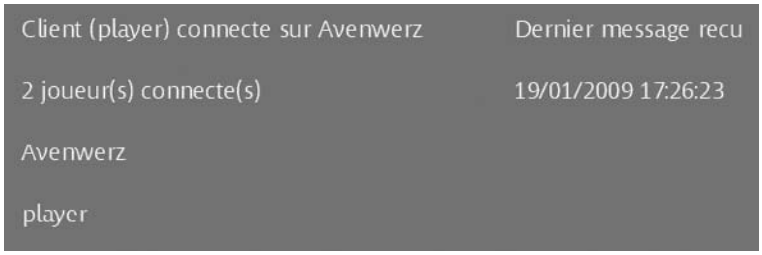


Figure 11-10

L'écran côté client

En résumé

Dans ce chapitre, vous avez découvert :

- comment créer un jeu en deux dimensions avec un effet de *scrolling* ;
- comment gérer des vues avec des objets de type `Viewport` pour créer un jeu multijoueur sur le même écran ;
- ce qu'est la plate-forme Windows Live ;
- comment fonctionnent les communications réseau avec XNA.

À la fin de ce chapitre, vous disposez de toutes les connaissances et notions nécessaires pour développer un jeu multijoueur en deux dimensions. Voyons à présent comment ajouter la troisième dimension/entrer dans la troisième dimension.