Introduction

In the previous part, you finally gained control over your spaceship. Using the arrow keys on the keyboard you are now able to navigate the spaceship anywhere you want.

This Silverlight application has a canvas of 1024 x 768 pixels as its RootVisual, which represents space. As it stands now, your spaceship is allowed to cross the boundaries of this canvas. In this part of the series you'll find out how you can make the spaceship reappear on the other side of the canvas.

Go ahead and download the code of part 3 and open it up in Visual Studio. If you haven't read the first three parts of this series, I encourage you to read those now.

Table Of Contents

Visual Studio 2010

Starting from this part, I'll develop the Asteroids game using Visual Studio 2010.

I just opened the solution and ran the Visual Studio Conversion Wizard. I also opted to target the .NET Framework 4. No changes were required in order to make the code we have so far compile and run.

You are free to continue developing in Visual Studio 2008 and target the .NET 3.5 (SP1) Framework if you want too. The code runs fine on both the .NET 3.5 or 4.0 Framework.

Remark: Although I'm using Visual Studio 2010 and the .NET 4 Framework this application still targets Silverlight 3.

Top of page

Parent Canvas

In part 2 of this series the GameElement type was introduced in the CGeers.Silverlight.GameEngine project. This type has a property called ParentCanvas. This property references the same instance as the GameSurface's Canvas property.

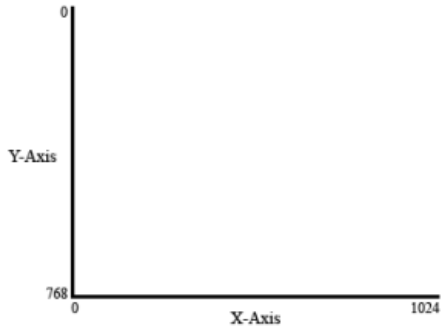
So each GameElement (sprite), such as the spaceship, already has a reference to the canvas in which it is contained. This is ideal, now we can add some common functionality to each sprite so that they all know how to stay within their ParentCanvas's bounds.

Top of page

X-Axis

In a 2-D game you only need to worry about two axes, namely the X and Y axis.

Figure 1 - X and Y axis



The canvas is 1024 x 786 pixels (width x height) in size. The X-Axis starts in the lower left corner at 0 and increases when you move to the right, all the way up to 1024.

You need to be able to determine if the sprite moved beyond the lower (0) and upper (1024) bounds of the X-Axis. Edit the Sprite type (Sprite.cs) found in the CGeers.Silverlight.GameEngine project and add the following code to it.

Listing 1 - Checking The X-Axis

```
public virtual bool HasMovedBeyondRightBound
{
    get
    {
        if ((this.X >= this.ParentCanvas.Width) ||
            ((this.X + this.Design.Width) >= this.ParentCanvas.Width))
        {
            return true;
        }
        return false;
    }
}

public virtual bool HasMovedBeyondLeftBound
{
    get { return this.X <= 0; }
}

public bool HasMovedBeyondXAxis
{
    get { return HasMovedBeyondLeftBound || HasMovedBeyondRightBound; }
}
```

The methods shown in Listing 1 are pretty self-explanatory.

- **HasMovedBeyondRightBound()**: Checks if the sprite has moved the upper bound of the X-Axis.
- **HasMovedBeyondLeftBound()**: Checks if the sprite has moved beyond the lower bound of the X-Axis.
- **HasMovedBeyondXAxis**: Checks if the sprite is positioned between the lower and upper bounds of the X-Axis.

When checking if the sprite is still positioned between the lower and upper bounds of the X-Axis it takes into account the width of the sprite (this.Design.Width).

Top of page

Y-Axis

If you recall your math lessons, you might expect that the Y-Axis starts in the lower left corner at zero and increases when you go up. However when referring to computer graphics the Y-Axis points down. Thus the value on the Y-Axis increases as you move down the screen (see Figure 1).

Thus the upper left corner can be expressed as a point with both it X and Y properties equalling zero (0,0). Add the following code to the Sprite class.

Listing 2 - Checking The Y-Axis

```
public virtual bool HasMovedBeyondUpperBound
{
    get { return this.Y <= 0; }
}

public virtual bool HasMovedBeyondBottomBound
{
    get
    {
        if ((this.Y >= this.ParentCanvas.Height) ||
            ((this.Y + this.Design.Height) >= this.ParentCanvas.Height))
        {
            return true;
        }
        return false;
    }
}

public bool HasMovedBeyondYAxis
{
    get { return HasMovedBeyondUpperBound || HasMovedBeyondBottomBound; }
}
```

The code to check whether the sprite has moved beyond the lower bound (0) or upper bound (768) of the Y-Axis is similar to the code displayed in Listing 1.

- **HasMovedBeyondUpperBound()**: Checks if the sprite has moved the upper bound of the Y-Axis.
- **HasMovedBeyondBottomBound()**: Checks if the sprite has moved beyond the lower bound of the Y-Axis.
- **HasMovedBeyondYAxis**: Checks if the sprite is positioned between the lower and upper bounds of the Y-Axis.

When checking if the sprite is still positioned between the lower and upper bounds of the Y-Axis it takes into account the height of the sprite (this.Design.Height).

Top of page

ISpriteDesign

When checking the position of the sprite compared to the X or Y axis, the width and height of the sprite in question are also taken into consideration. The sprite's dimensions are accessed through the ISpriteDesign interface.

Add Width and Height properties to the ISpriteDesign interface found in the CGeers.Silverlight.GameEngine project.

Listing 3 - ISpriteDesign

```
public interface ISpriteDesign
{
    UIElement UIElement { get; }
    double Width { get; set; }
    double Height { get; set; }
}
```

Since each sprite's design will be implemented using the UserControl type you don't need to worry about implementing these new properties. The UserControl type already has these. By adding them to the ISpriteDesign interface they are now accessible in the Sprite class.

Top of page

Bounds Of The Parent Canvas

By adding the methods of Listing 1 and 2 to the Sprite class you can easily check if a sprite is crossing the bounds of either the X or Y Axis. Just add the following method to the Sprite class.

Listing 4 - HasMovedOutOfParentCanvasBounds() Method

```
public bool HasMovedOutOfParentCanvasBounds
{
    get { return HasMovedBeyondXAxis || HasMovedBeyondYAxis; }
}
```

By calling the KeepSpriteWithinParentCanvasBounds() method you can make the sprite reappear on the other side if it goes off the canvas in either direction (left, right, top, bottom).

Let's wrap up the changes to the Sprite class for this part. Add the following method to the Sprite class.

Listing 5 - KeepSpriteWithinParentCanvasBounds() Method

```
protected void KeepSpriteWithinParentCanvasBounds()
{
    if (this.HasMovedBeyondRightBound)
    {
        this.X = 1;
    }
    else if (this.HasMovedBeyondLeftBound)
    {
        this.X = this.ParentCanvas.Width - this.Design.Width;
    }
    if (this.HasMovedBeyondBottomBound)
    {
        this.Y = 1;
    }
    else if (this.HasMovedBeyondUpperBound)
    {
        this.Y = this.ParentCanvas.Height - this.Design.Height;
    }
}
```

Top of page

Update The Sprites

As you can see each sprite does not automatically check its position compared to the bounds of the GameSurface's canvas. I've chosen an opt-in approach for this. Sometimes you want a sprite to reappear on the other side, at other times you want to dispose the sprite when it moves beyond these bounds (e.g. bullets) and sometimes you even want to allow that a sprite can move beyond them.

Just let each individual sprite decide what to do. Open up the Spaceship class found in the Asteroids project and override its Update() method as shown below.

Listing 6 - Spaceship Update() Method

```
public override void Update(TimeSpan elapsedTime)
```

```
{
    this.KeepSpriteWithinParentCanvasBounds();
}
```

Now you just need to add some code to make sure that each Sprite's Update() method is called when a frame needs to be rendered. Sounds like a task for the GameSurface class. Add the following line of code to the GameSurface's GameLoopTick(...) method just before you trigger the RenderFrame event.

Listing 7 - GameSurface RenderFrame() Method

```
this._gameElements.ForEach(gameElement => gameElement.Update(eventArgs.ElapsedTime));
```

Now each sprite is notified when a new frame needs to be rendered. You can update each sprite within its own Update(...) method to keep things nicely separated. This will surely come in handy for later parts when we add other sprites such as asteroids, bullets...etc.

Top of page

Summary

Start the application and navigate your spaceship anywhere you want. Whatever you do, the spaceship will stay within the bounds of the GameSurface's canvas!

If you cross the boundaries of the canvas, you'll notice that the spaceship will reappear on the other side. In the next part I'll show you, how you easily add asteroids to the game. All the code to enable this, is already in place.

Most sprites should be confined to the bounds of the canvas which they inhabit. In this part of the Asteroids series you've learned how you can easily achieve this goal by applying two steps:

- Checking the sprite's position compared to the canvas' X and Y - axis bounds
- Updating the position of the sprite if it crosses these boundaries

This code will surely come in handy when adding asteroids to the game as these sprites are also not allowed to cross the boundaries of the GameSurface's canvas.

Any comments, questions, tips…etc. are always appreciated.

Top of page

Download

You can find the source code for this article on the Download page of this blog.

Top of page