SPRITES – Keeping Them On The Screen

In the last lesson we learned how to move sprites around on the screen. One problem we ran into was moving the sprite off the screen so it can't be seen any more.

Can you think of a situation in which you want this behavior? Sure! What about a Frogger type game in which objects appear on one side of the screen, move across and off the other side?

On the other hand, there are certainly situations in which you don't want a sprite to be able to move off-screen (the paddles in a pong type game), or perhaps you want it to reappear on the other side of the screen if it goes off (your ship in an asteroids type game).

This all boils down to screen bounds management. To do this we need to be able to:

1. Determine how big the visible portion of the GUI screen is
2. Determine how big our sprite is
3. Determine where our sprite is on the screen (we learned this last lesson)
4. Limit or control the positioning of our sprite on the screen

Let's get to it…

SPRITES – Keeping Them On The Screen

Let's continue working with the program we wrote for the last lesson ***IntroToSprites***. If you can't find it … well … go back to the lesson and recreate it! Sorry!

If you recall, the last thing we did was add code in the ***KeyDown*** event handler to adjust the location point of our sprite.

We're going to add some code to that event handler that checks if our sprite has hit an edge of the screen. So, how do we tell where the edges of the screen are?

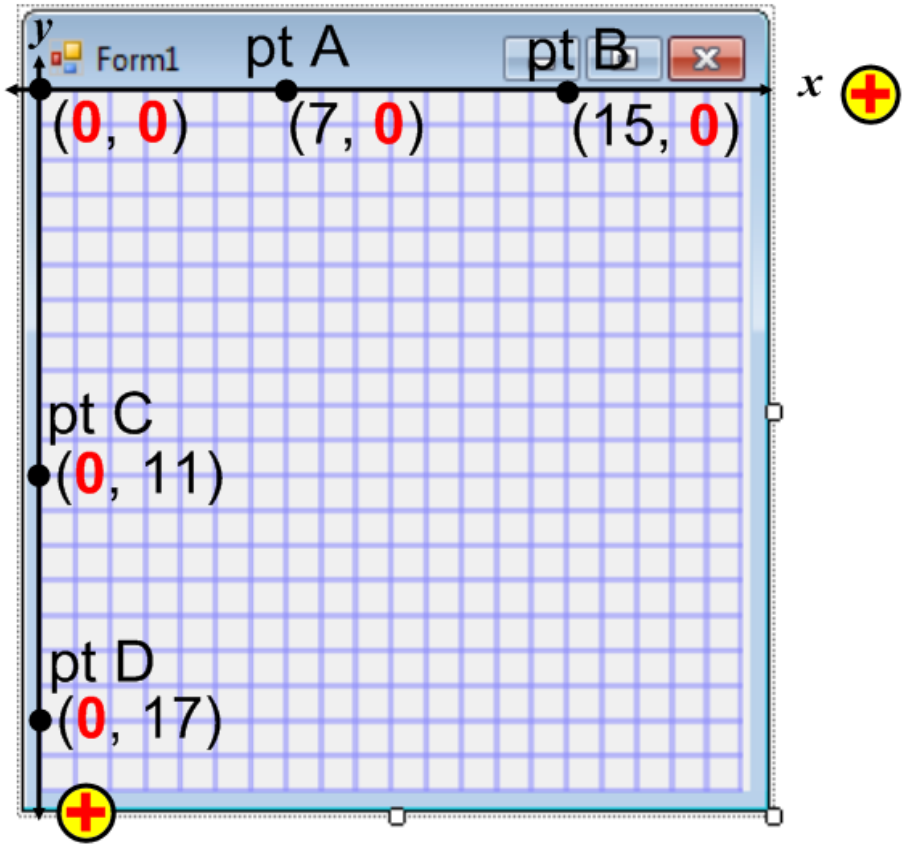
Remember the screen has a coordinate system with ***x*** and ***y*** coordinates. The top-left corner of the screen is the origin, or point (0, 0). The ***x*** coordinate gets bigger to the right and the ***y*** coordinate gets bigger going down (this is reversed from the normal Cartesian coordinate system).

Let's start with the easy edges…

How can we tell if our sprite is at the top edge of the screen? Hmm…well, the top edge is an up/down edge so that would be related to the ***y*** coordinate. What is the ***y*** coordinate of the top of the screen? Since the origin as at the top-left corner and it is point (0, 0), that would mean that anywhere along the top edge of the screen is ***y*** coordinate 0.

Okay, that was easy, what about the left edge? Same deal isn't it? The left edge is a left/right edge so we're talking about the ***x*** coordinate. Again, the origin is at the top-left corner so anywhere along the left edge the ***x*** coordinate will be 0.

Here are some example points to illustrate. Pts A and B are on the top edge, C and D on the left.



What about the bottom and left edges? These are harder because their coordinates will change with the size of the window. So in order to know where the right and bottom edges are, we need to know the size of the screen.

This isn't as obvious as you'd think.

But wait you say…isn't there a size property for the form? Can't we use that to tell how big the screen is? Nope, sorry … the size property is for the entire form ***including*** the title bar and borders of the window. What we're concerned about is the light gray portion of the window. If you search through all the properties, you won't find any others listed that can help us.

This is where the built-in help comes in. If you bring up the help for form (in the code window put your cursor in the word ***form*** on the `public partial class Form1 : Form` line, then hit F1), you will see there are many, many properties listed that are not listed in the Visual C# properties panel. So many in fact, it can be a bit overwhelming. Be that as it may, if you are diligent and look hard enough, you will find the ***ClientSize*** property which has the following description:

> *Gets or sets the size of the client area of the form*

Go to the detail for this property and do some more reading…you'll learn this means

> *The size of the client area of the form is the size of the form **excluding** the borders and the title bar … You can use this property to get the proper dimensions when performing graphics operations…*

Excellent! This is exactly what we want!

SPRITES – Keeping Them On The Screen

Okay, now what? How can we tell if our sprite has hit the right edge of the screen?

Well, let's think about this … let's assume we don't want any part of our sprite to go off screen. This means we don't want the ***right edge of our sprite*** to go past the right edge of the screen.

How do we figure out where the right edge of our sprite is? All we need is a little bit of 5th grade math and we have it! We can easily figure out where the left edge of our sprite is … the sprite's location point is at the top-left corner of the sprite so the ***x*** coordinate is at the left edge of our sprite. Where is the right edge? To determine that, all we need to know is how wide our sprite is … which is very easy; remember we named our sprite ***spritePixBox***?

```
spritePixBox.Width
```

… is the width of our sprite.

The ***x*** coordinate of the right edge of the screen is

```
this.ClientSize.Width
```

… which makes sense because if you start that left at 0 and add the width you'll get the coordinate of the right screen edge … except what the heck is the `this.` part all about?

Remember, all this code is inside our ***form*** class which encapsulates (represents and contains) our GUI. When we are asking for the ***ClientSize*** width of the GUI, we're asking for the width of the form. Since we're "inside" the form, we can refer to the form (the class) we're in as `this`. Notice in the code window, it is in dark blue text, and dark blue text in the code window means a C# keyword. ***this*** is a reserved keyword in C# that refers to the class you are in.

The ***y*** coordinate of the bottom edge of the screen is

```
this.ClientSize.Height
```

and the height of the our sprite is

```
spritePixBox.Height
```

Do the math now … the right edge of our sprite is the ***x*** coordinate of the sprite's left edge plus the sprite's width:

```
spritePixBox.Location.X + spritePixBox.Width
```

The bottom edge of our sprite is the ***y*** coordinate of the sprite's top edge plus the sprite's height:

```
spritePixBox.Location.Y + spritePixBox.Height
```

Yup, a 5th grader could do that!

SPRITES – Keeping Them On The Screen

Remember in our code from last lesson, we had the adjusted (new) location point for our sprite in a ***Point*** variable named ***tempPoint***?

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    Point tempPoint = spritePixBox.Location;    // create temporary Point variable

    if (e.KeyCode == Keys.A)                    // was the A kit hit?
    {
        tempPoint.X -= 1;                       // yup...move one to the left
    }

    if (e.KeyCode == Keys.S)                    // was the S kit hit?
    {
        tempPoint.Y += 1;                       // yup...move one down
    }

    if (e.KeyCode == Keys.D)                    // was the D kit hit?
    {
        tempPoint.X += 1;                       // yup...move one to the right
    }

    if (e.KeyCode == Keys.W)                    // was the W kit hit?
    {
        tempPoint.Y -= 1;                       // yup...move one up
    }

    spritePixBox.Location = tempPoint;          // copy the temp Point into the sprite location point
}
```

In the last line we're putting the adjusted location point back into the sprite, which actually makes it move on the screen. Wouldn't it be great if we could make sure the new point would keep the sprite on the screen before we put it back into the sprite?

Makes sense to me! ☺ All we need to do is add code just before that last line that checks to see if we're moving off screen. If we are, then all we need to do is adjust the new point back a bit so the sprite stays right at the edge of the screen. Hmm, more math… ☺

SPRITES – Keeping Them On The Screen

We'll need to handle each screen edge one at a time. Let's do the easy ones first…

Top screen edge

If the ***y*** coordinate of the sprite location point is less than zero,
Then reset it to zero.

Left screen edge

If the ***x*** coordinate of the sprite location point is less than zero,
Then reset it to zero.

And now the harder ones…

Bottom screen edge

Add the height of the sprite to the ***y*** coordinate of the sprite location point.
If this is greater than the height of the screen,
Then reset the ***y*** coordinate to the screen height minus the sprite height.

Right screen edge

Add the width of the sprite to the ***x*** coordinate of the sprite location point.
If this is greater than the width of the screen,
Then reset the ***x*** coordinate to the screen width minus the sprite width.

I'll code the top and right edges for you; you get to figure out how to code the bottom and left edge. The picture is of the bottom of the ***KeyDown*** event handler…

```
    if (e.KeyCode == Keys.D || e.KeyCode == Keys.Right)                    // was the D kit hit?
    {
        tempPoint.X += 1;                       // yup...move one to the right
    }

    if (e.KeyCode == Keys.W || e.KeyCode == Keys.Up)                    // was the W kit hit?
    {
        tempPoint.Y -= 1;                       // yup...move one up
    }

    // Make sure the sprite stays on the screen...
    // ...is the top edge of the sprite above the top edge of the screen?
    // ...we have the new position/location in tempPoint, check that before we update the sprite
    if (tempPoint.Y < 0)
    {
        // ...yup...reset the sprite location Y to make sure the top of the sprite stays on the screen
        tempPoint.Y = 0;
    }

    // ...is the right edge of the sprite beyond the right edge of the screen?
    if (tempPoint.X + spritePixBox.Width > this.ClientSize.Width)
    {
        // ...yup...reset the sprite location X to make sure the right of the sprite stays on the screen
        tempPoint.X = this.ClientSize.Width - spritePixBox.Width;
    }

    spritePixBox.Location = tempPoint;          // copy the temp Point into the sprite location point
}
```

SPRITES – Keeping Them On The Screen

Your turn! Try this to make sure it works, and then …

Your tasks

1. Add the code to keep the sprite from going off the left and bottom edges of the screen. ☺

2. Make a version of this that causes the sprite to wrap around to the other side of the screen. As an example, when the ***left edge of the sprite*** goes off the ***right edge of the screen***, make the sprite appear on the left side of the screen.