Bouncing animation in C#.

This tutorial will show you how to create a bouncing animation in C#. This is a very fun tutorial because it can be recreated into anything you want.

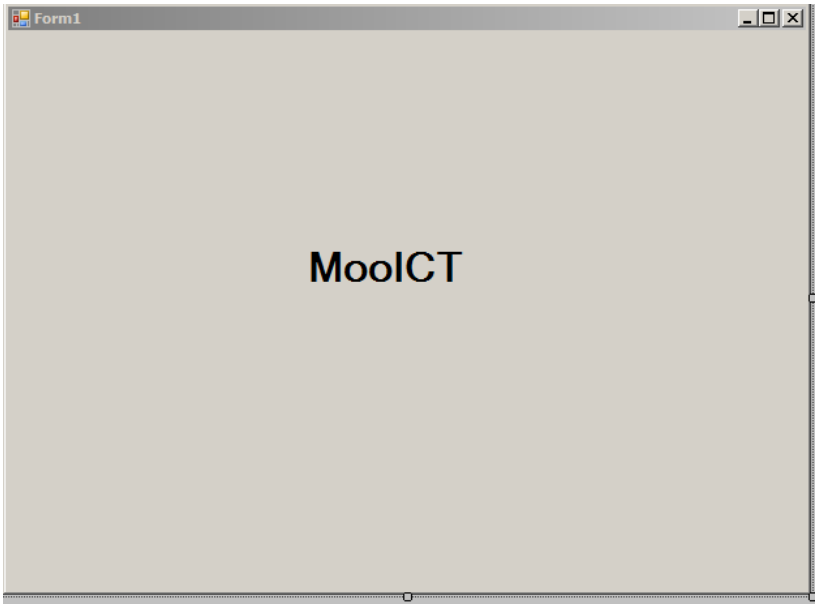
This is what we are doing in this tutorial, we will create a border bouncing animation in c# with a label. Each time it hits a border we will change the colour of the label.

Lets get started. Create a new project in visual studio and call it borderbounce.

Drag a Label to the screen and make it large so you can see it better.

drag a timer component to the form too.

This is what we have so far.



Now double click on the form to be exported into the code view. Visual studio will also create a form load function for us. We will need to include some code in there shortly.

This is the code view at the moment.

```
public partial class Form1 : Form
{



    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {


    }

}
```

Before the form1() function we can add some global variables. Enter the four variables below:

private int objWidth; this variable will hold the width of the label from the form. This will be defined later so the value is empty for now.

private const int objHeight = 50; this constant variable will hold the height of the label. Good to be genereous with height.

private int objX, objY; // Position. There two variables will hold the x and y position for the label.

private int volX, volY; // Velocity. these two variables will manipulate the speed of the x and y positions.

Inside the form load function lets add the following code.

```
public partial class Form1 : Form
{

    private int objWidth;
    private const int objHeight = 50;
    private int objX, objY;   // Position.
    private int volX, volY; // Velocity.

    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {

        // Pick a random start position and velocity.
        Random rnd = new Random();
        volX = rnd.Next(1, 4);
        volY = rnd.Next(1, 4);
        objX = label1.Left;
        objY = label1.Top;
        objWidth = label1.Width;

        objX = rnd.Next(0, ClientSize.Width - objWidth);
        objY = rnd.Next(0, ClientSize.Height - objHeight);

        // Use double buffering to reduce flicker.
        this.SetStyle(
            ControlStyles.AllPaintingInWmPaint |
            ControlStyles.UserPaint |
            ControlStyles.DoubleBuffer,
            true);
        this.UpdateStyles();

    }
```

// Pick a random start position and velocity.

Random rnd = new Random(); We are calling a random class and storing it inside the rnd variable.

volX = rnd.Next(1, 4); Volecity x will contain a random number between 1 and 4

volY = rnd.Next(1, 4); volecity y will contain a number between 1 and 4.

objX = label1.Left; objx will contain the left value of label 1.

objY = label1.Top; objy will contain the top value of label 1.

objWidth = label1.Width; obj width will calculate and store the width of label 1.

objX = rnd.Next(0, ClientSize.Width - objWidth); We will set the x position of the label to a random place in the form.

objY = rnd.Next(0, ClientSize.Height - objHeight); We will set the y position of the label to a random place in the form.

```
// Use double buffering to reduce flicker.
this.SetStyle(
    ControlStyles.AllPaintingInWmPaint |
    ControlStyles.UserPaint |
    ControlStyles.DoubleBuffer,
    true);
this.UpdateStyles();
```

This highlighted line in the top is to set the style of the animation so it doesn't flicker on the screen. As you can tell we will use a timer to animate the object on the screen. Each nano second it will move a pixel, often times c# cannot render smooth animation. We are using this small hack to reduce the flicker to show smoother animation.

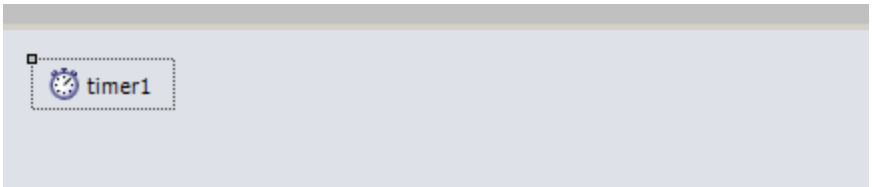
For you tech heads here is a technical explanation:

from Microsoft:

The documentation for OptimizedDoubleBuffer indicates that it will help to reduce flicker and has the same effect as setting the UserPaint and AllPaintingInWmPaint ControlStyles to true. However, if the OptimizedDoubleBuffer ControlStyle is set to true without setting UserPaint and AllPaintingInWmPaint there is no visible reduction of flickering. Setting the UserPaint and AllPaintingInWmPaint ControlStyles will result in the same reduced flicker optimisation as in Visual Studio 2003 when setting the DoubleBuffer, UserPaint and AllPaintingInWmPaint ControlStyles. In other words - there doesn't seem to be any difference between OptimizedDoubleBuffer and the obsolete DoubleBuffer.

Ok bla bla bla. We get it moving on.

Now lets go back to the design view and double click on tiem timer 1 component which was added earlier.



once we double click it, visual studio will drop us back to the code view.

```
private void timer1_Tick(object sender, EventArgs e)
{

}
```

Bottom of the code this line is added now. This event will trigger each second. We will import all our animation log inside the function.

```
private void timer1_Tick(object sender, EventArgs e)
{
    label1.Left = objX;
    label1.Top = objY;

    objX += volX;
    if (objX < 0)
    {
        volX = -volX;
        Random rnd = new Random();
        label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255));
    }
    else if (objX + objWidth > ClientSize.Width)
    {
        volX = -volX;
        Random rnd = new Random();
        label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255));
    }

    objY += volY;
    if (objY < 0)
    {
        volY = -volY;
        Random rnd = new Random();
        label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255));
    }
    else if (objY + objHeight > ClientSize.Height)
    {
        volY = -volY;
        Random rnd = new Random();
        label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255));
    }

    Refresh();

}
```

label1.Left = objX; we are setting up the left of label 1 to obj x

label1.Top = objY; we are setting up top of label 1 to objy

These both variables will change dynamically when the time is running.

objX += volX; we are using a += which means the X direction of label 1 will increase each second.

We want the object to bounce from the edges and on to the other direction. Use this if statement to check that

if (objX < 0)

if the objects x is less than zero meaning if its leaving the form then we can change its direction

volX = -volX;

See how its changed to = -volX its negative now.

We also want to change the colour of the label each time it bounces off the edges. Check the code below

Random rnd = new Random(); Calling the random class and storing it inside the rnd vairable.

label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255));

In the line above we are changing the colour using the random numbers next function. Its selecting colours from all the RGB elements.

```
else if (objX + objWidth > ClientSize.Width)
{
   volX = -volX;
   Random rnd = new Random();
   label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255));
}
```

In the code above we are doing the same thing as before except we checking the right side of the screen this time. We are measuring is the label is leaving the right side of the screen then we are changing its speed to negative so it bounces off the edge to the other side.

```
objY += volY;
if (objY < 0)
{
   volY = -volY;
   Random rnd = new Random();
   label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255));
}
else if (objY + objHeight > ClientSize.Height)
{
   volY = -volY;
   Random rnd = new Random();
```

```
            label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255));
        }
```

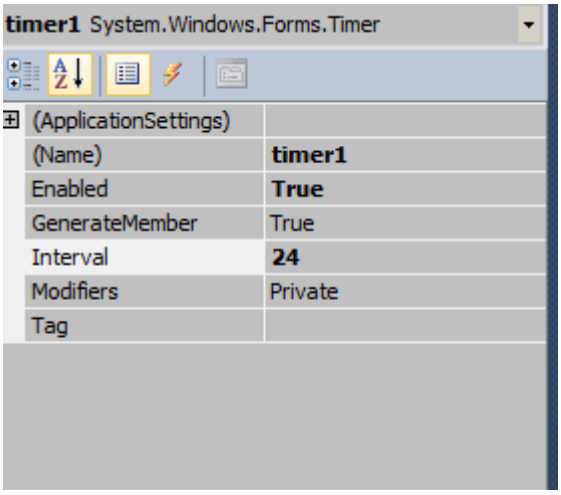
We have done the same to the Y direction for the label as for the X direction.

Its all worked up right now.

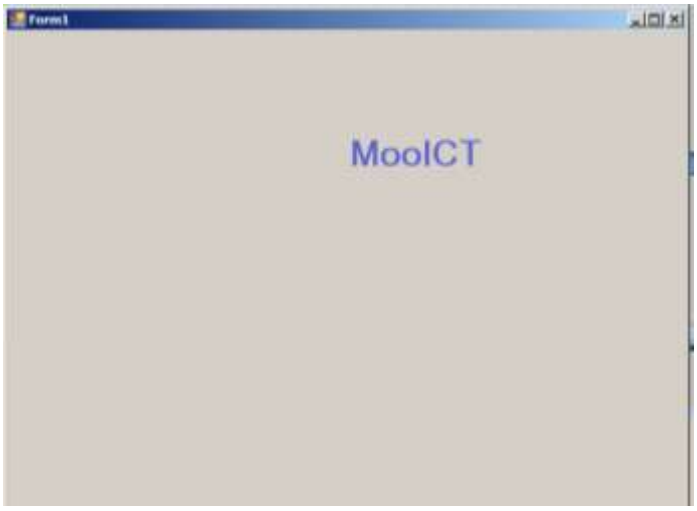
lastly we are going to call the Refresh() function inside the timer. This function will refresh the form each time the timer ticks so there is less lagging on the screen.

If you run the program now it will not do anything. Why not? We set everything up so far correct? Nope we haven't done everything yet.

We need to enable the timer so it runs once the program has started.



Here is the settings for the timer1 properties. Change them and run the program.



As you can see its working perfectly.

If you want to use a picture box and bounce it around you can do that too.

Try setting up a picture of instead of a label and see how that works out.

Here is the full source code for the app.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace bouncingBall
{
    public partial class Form1 : Form
    {

        private int objWidth;
        private const int objHeight = 50;
        private int objX, objY;   // Position.
```

```
        private int volX, volY; // Velocity.

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

            // Pick a random start position and velocity.
            Random rnd = new Random();
            volX = rnd.Next(1, 4);
            volY = rnd.Next(1, 4);
            objX = label1.Left;
            objY = label1.Top;
            objWidth = label1.Width;

            objX = rnd.Next(0, ClientSize.Width - objWidth);
            objY = rnd.Next(0, ClientSize.Height - objHeight);

            // Use double buffering to reduce flicker.
            this.SetStyle(
                ControlStyles.AllPaintingInWmPaint |
                ControlStyles.UserPaint |
                ControlStyles.DoubleBuffer,
                true);
            this.UpdateStyles();

        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            label1.Left = objX;
            label1.Top = objY;

            objX += volX;
            if (objX < 0)
            {
                volX = -volX;
                Random rnd = new Random();
                label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255),
rnd.Next(255));
            }
            else if (objX + objWidth > ClientSize.Width)
            {
                volX = -volX;
                Random rnd = new Random();
                label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255),
rnd.Next(255));
            }

            objY += volY;
            if (objY < 0)
            {
                volY = -volY;
                Random rnd = new Random();
                label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255),
rnd.Next(255));
            }
            else if (objY + objHeight > ClientSize.Height)
            {
                volY = -volY;
```

```
                Random rnd = new Random();
                label1.ForeColor = Color.FromArgb(rnd.Next(255), rnd.Next(255),
rnd.Next(255));
            }

            Refresh();

        }

    }
}
```