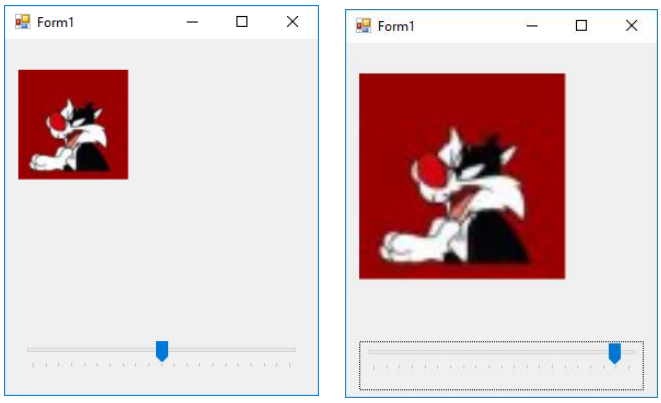
Bouncing Shapes

Task 1: The TrackBar Control

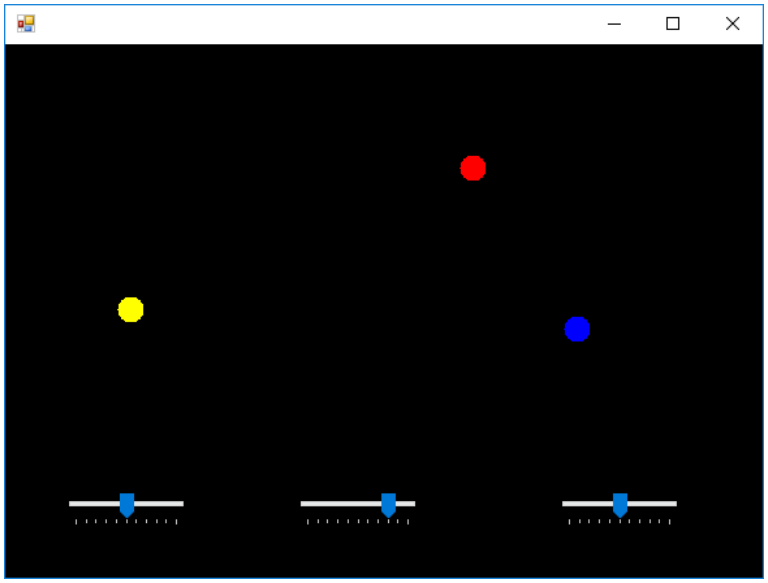
1. Start a new project.
2. Place a *PictureBox* control and a *TrackBar* control on your Form.
   The *TrackBar* has a *Value* property that has a minimum and maximum amount. In my example, the minimum is set to 0 (far left) and the maximum to 200 (far right).
   Also investigate the *TrackBar Frequency* property.
3. Write a *trackBar1_ValueChanged* handler so that, as the track bar changes in value, the height and width of the image changes by the same amount.



Task 2: Bouncing Shapes

********* CHECKPOINT 6 *********

Many games, simulations and screen savers involve objects that move around the screen and bounce off the edges.



For this exercise, create an application that has three geometric shapes that move horizontally and vertically about the screen. When a ball hits the edge, it bounces off in the reverse direction.
A suggested approach to solving this problem:

Decomposition

Break down the problem:

- Describe the problem in in 1-2 sentences.
  This should be copied to the **Purpose** section of your **Block Comment**.
- Describe the game, by explaining the behaviour of each component in the game.
  Each ball…..
  The trackbars….

This should be copied to the **Description** section of your **Block Comment**.

Planning the first iteration of your solution

Where will you start?

- What is your MVP (minimum viable product)? Remember to start small, abstract only what is needed to build the MVP (and no more).
- How many classes do you need? What will each class be named?
- List the behaviours that you need for your class. These are the methods.
- From the methods, deduce the data that you need to store in the fields.
- Draw **UML class diagrams**.
- Draw up a **UML sequence diagram** to check how the objects will interact in the timer1_Tick() event.

Build your MVP

Code up your MVP.

- Make sure it compiles!
- You may need to update your UML diagrams?

Iterative refinement

As you add new features:

- Update your **UML class diagrams**.
- Identify the purpose of each class? This is your **class comment.**
- Remember that a method should do only one thing. Do you need to break your methods into sub-methods?
- State the purpose of each method in one short sentence. This becomes the **method comment**.

.

Double Buffering

If you draw to a graphics object piece by piece, you will get an irritating flicker.
To reduce the flicker, you can double buffer the drawing process. You draw all images to a graphics object that sits on an image in memory. When all drawing has been completed, you draw from this buffered image to screen (our graphics object) in a single operation.

```
public partial class Form1 : Form
{
    private Bitmap bufferImage;
    private Graphics bufferGraphics;
    private Graphics graphics;
……….



public Form1()
    {
        InitializeComponent();

        // off-screen image, stored in memory (buffer)
        bufferImage = new Bitmap(Width, Height);

        // drawing surface that sits on the bufferImage
        bufferGraphics = Graphics.FromImage(bufferImage);

        graphics = CreateGraphics();


        // remember to pass the bufferGraphics to the World/Controller object
        // All drawing is made to this bufferGraphics drawing surface
        world = new World (bufferGraphics,…..)
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        bufferGraphics.FillRectangle(Brushes.Black, 0, 0, Width, Height);
        world.Run();
        graphics.DrawImage(bufferImage, 0, 0);
    }
```

Extra Functionality

Add **extra functionality** to improve your solution. Some ideas could be:

Add sound each time a ball bounces off an edge.
Change your shapes to rectangles.
Add more balls, delete some balls.
Randomly change the colours of the balls.
Add extra methods to the Ball class, to the World class.

Remember to refer to these in your Block Comment.
