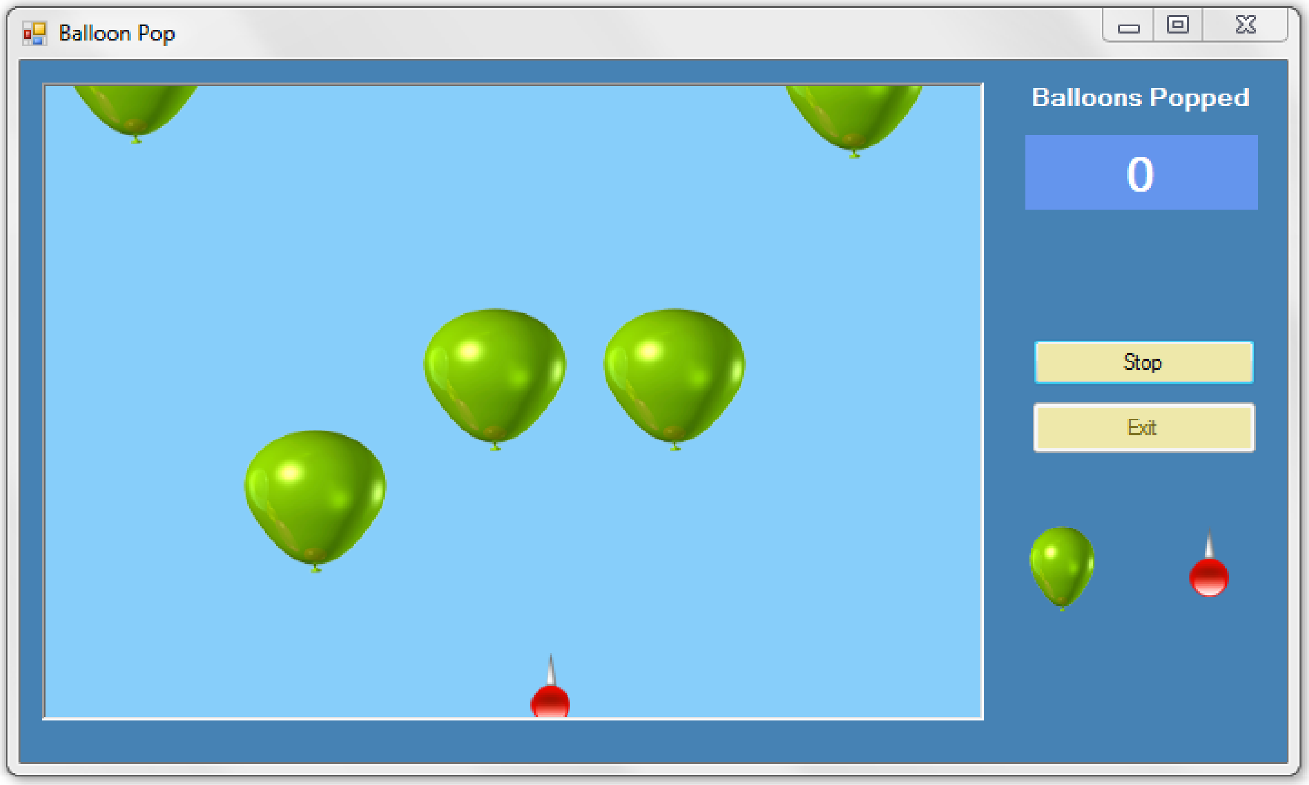
# Project 02 – Balloon Pop

Colorful balloons are dropping from the sky. You maneuver your popping device under them to make them pop and get a point. You try to pop as many balloons as you can in 30 seconds.

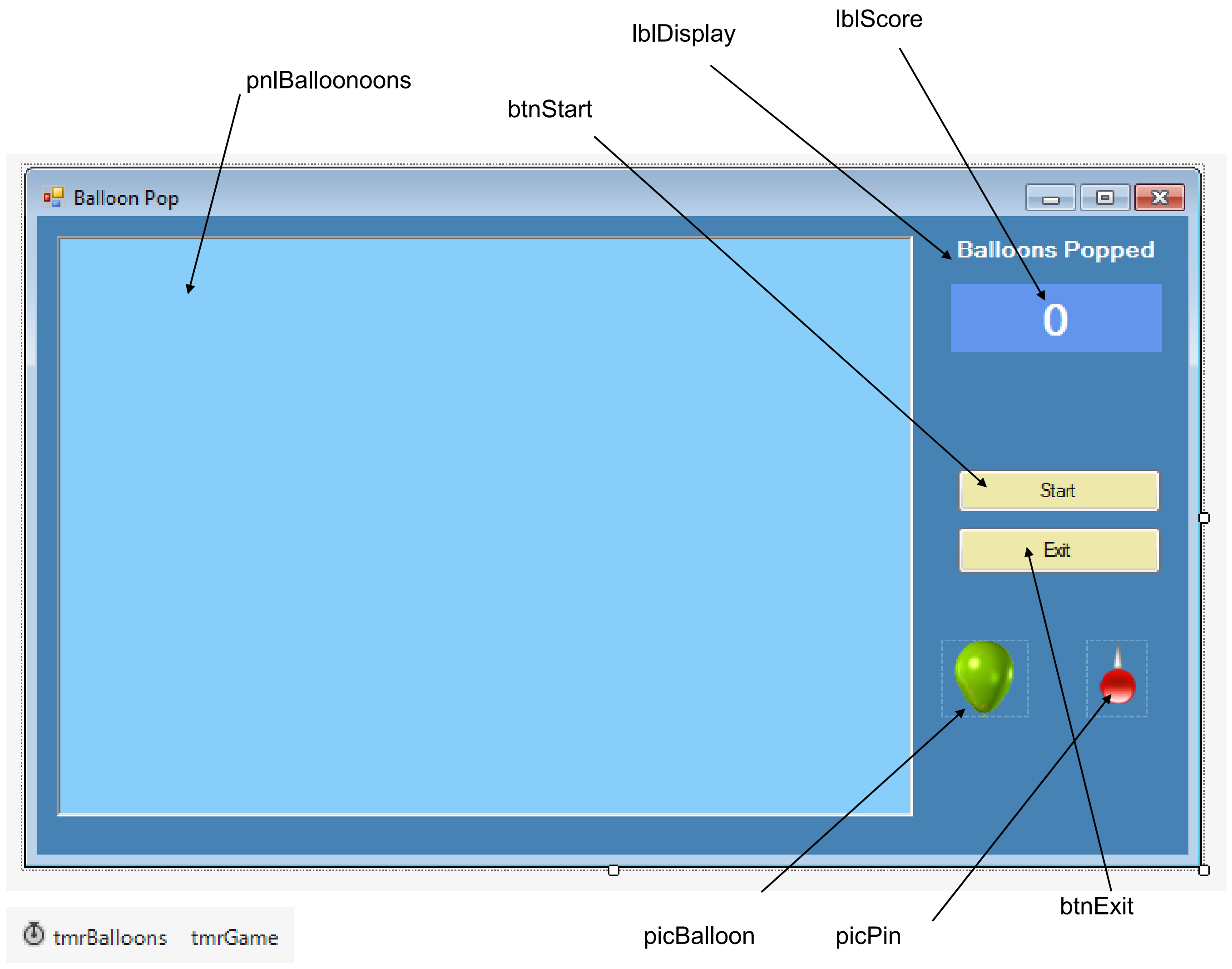
## Project Design

All of the game action will go on in a panel control. There will be five possible balloons, the image used is contained in a picture box. A picture box control will also hold the "pin" image. This image will be moved using keys on the keyboard. A button will control starting and stopping the game. Another button will exit the program. The current score (number of balloons popped) will be displayed in a label.



## Place Controls on Form

When done setting properties, my form looks like this:



## Setting Control Properties

**Form1** Form:

| **Property Name** | **Property Value** |
|---|---|
| FormBorderStyle | FixedSingle |
| StartPosition | CenterForm |
| KeyPreview | True (this allows us to detect key presses) |

**panel1** Panel:

| **Property Name** | **Property Value** |
|---|---|
| Name | pnlBalloonoons |
| BackColor | Light Blue |
| BorderStyle | FixedSingle |

**timer1** Timer:

| **Property Name** | **Property Value** |
|---|---|
| Name | tmrBalloons |
| Interval | 100 |

**timer2** Timer:

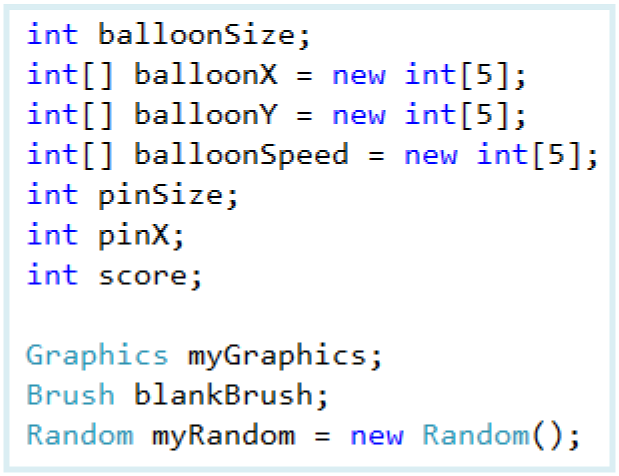
| **Property Name** | **Property Value** |
|---|---|
| Name | tmrGame |
| Interval | 30000 |

## Write Event methods

The Balloon Pop game is simple, in concept. To play, click the **Start** button. Five balloons will drop down the panel, each at a different speeds. Use the keyboard to move the pin. If the pin is under a balloon when a collision occurs, the balloon pops and you get a point. Balloons reappear at the top after popping or after reaching the bottom of the screen without being popped. You pop as many balloons as you can in 30 seconds. At that point, a ‘Game Over’ message appears. You can click **Start** to play again or click **Exit** to stop the program.

It looks like there are only three events to code, clicking the **Start** button, clicking the **Exit** button, or using **frm_KeyDown** to check for pin key presses. But, recall there are two timer controls on the form. The control named **tmrBalloons** controls the balloon animation, updating the panel 10 times a second (Interval is 100). The timer control named **tmrGame** controls the overall time of the game. It generates a Tick event only once - when the game is over (Interval is 30000 - that’s 30 seconds). So, in addition to button clicks and key down events, we need code for two Timer events. There is a substantial amount of C# code to write here, even though you will see there is a lot of repetition. We suggest writing the event methods in stages. Write one method or a part of a method. Run the project. Make sure the code you wrote works. Add more code. Run the project again. Make sure the added code works. Continue adding code until complete. Building a project this way minimizes the potential for error and makes the debugging process much easier.

Each balloon will occupy a square region. We will compute the size (balloonSize) of the balloon to fit nicely on the panel. We need array variables to keep track of each balloon's location (balloonX, balloonY) and dropping speed (balloonSpeed). We need to know the pin's size (pinSize) and position (pinX). We also need a graphics object to draw the balloons (myGraphics) and a blank brush object (blankBrush, for erasing balloons). We need a variable to keep track of the score. Lastly, we need a random number object (myRandom). Add this code to the **general declarations** area:



```
int balloonSize;
int[] balloonX = new int[5];
int[] balloonY = new int[5];
int[] balloonSpeed = new int[5];
int pinSize;
int pinX;
int score;

Graphics myGraphics;
Brush blankBrush;
Random myRandom = new Random();
```

The array **balloonSpeed** holds the five speeds, representing the number of pixels a balloon will drop with each update of the viewing panel. We want each balloon to drop at a different rate. In code, each speed will be computed using:

```
myRandom.Next(3,7);
```

Or, it will be a random value between 3 and 6. A new speed will be computed each time a balloon starts its trip down the panel. How do we know this will be a good speed, providing reasonable dropping rates? We didn't before the project began. This expression was arrived at by 'trial and error.' We built the game and tried different speeds until we found values that worked. You do this a lot in developing games. You may not know values for some numbers before you start. So, you go ahead and build the game and try all kinds of values until you find ones that work. Then, you build these numbers into your code.

Use this **Form1_Load** method:

```
private void frmPin_Load(object sender, EventArgs e)
{
    int x;
    // Have the balls spread across the panel with 20 pixels borders
    balloonSize = (int)((pnlBalloon.Width - 6 * 20) / 5);

    x = 10;
    for (int i = 0; i < 5; i++)
    {
        balloonX[i] = x;
        x = x + balloonSize + 20;
    }
    // Make pin one-half the balloon size
    pinSize = (int)(balloonSize / 2);

    myGraphics = pnlBalloon.CreateGraphics();
    blankBrush = new SolidBrush(pnlBalloon.BackColor);
    // Give form focus
    this.Focus();
}
```

In this code, initial horizontal positions for each of the balloons are computed (balloonX array). The balloons are spread evenly across the panel (see if you can understand the code). The pin is made to be one-half the balloon size (pinSize). Lastly, the graphics object and brush object are created and the form is given focus so KeyDown events can occur.

Create then add this code to the **Form1_FormClosing** event to dispose of our objects:

```
private void picPin_FormClosing(object sender, FormClosingEventArgs e)
{
    myGraphics.Dispose();
    blankBrush.Dispose();
}
```

To move the pin (using DrawImage), we need a **Form1_KeyDown** event method (the panel control does not have a KeyDown event). Make sure you have set the form's **KeyPreview** property to **True**, so the KeyDown event will be "seen." Pick a key that will move the pin to the left and a key that will move it to the right. I chose **A** for **left** movement and **F** for **right** movement.

```
private void frmPin_KeyDown(object sender, KeyEventArgs e)
{
    // Erase arrow at old location
    myGraphics.FillRectangle(blankBrush, pinX,
        pnlBalloon.Height - pinSize, pinSize, pinSize);
    // Check for left and right keys and compute arrow position
    if (e.KeyCode == Keys.A)
    {
        pinX = pinX - 5;
    }
    else if (e.KeyCode == Keys.F)
    {
        pinX = pinX + 5;
    }
    // Position arrow
    myGraphics.DrawImage(picPin.Image, pinX,
        pnlBalloon.Height - pinSize, pinSize, pinSize);
}
```

Notice if the A key is pressed, the pin (**imgPin**) is moved to the left by 5 pixels. The pin is moved right by 5 pixels if the F key is pressed. Again, the 5 pixels value was found by 'trial and error' - it seems to provide smooth motion. After typing in this method, save the project, then run it. Make sure the pin moves as expected. Press the A key to see it. It should start at the left side of the form (pinX = 0) since we have not given it an initial position. This is what we meant when we suggested building the project in stages. Notice there is no code that keeps the pin from moving out of the panel - you could add it if you like.

The **btnExit_Click** method is simple, so let's get it out of the way first.

```
this.Close();
```

Let's outline the steps involved in the **btnStart_Click** event. We use this button for two purposes. It either starts the game (**Text** is **Start**) or stops the game (but, not the program - **Text** is **Stop**). So, the Click event has two segments. If Text is Start, the steps are:

- Set btnStart Text to "Stop"
- Disable btnExit button
- Clear balloons off screen
- Set score to 0
- Initialize each balloon's position and speed
- Initialize pin position
- Give form focus (so KeyDown can be recognized)
- Start the timers

If the Text is Stop when the button is clicked, the program steps are:

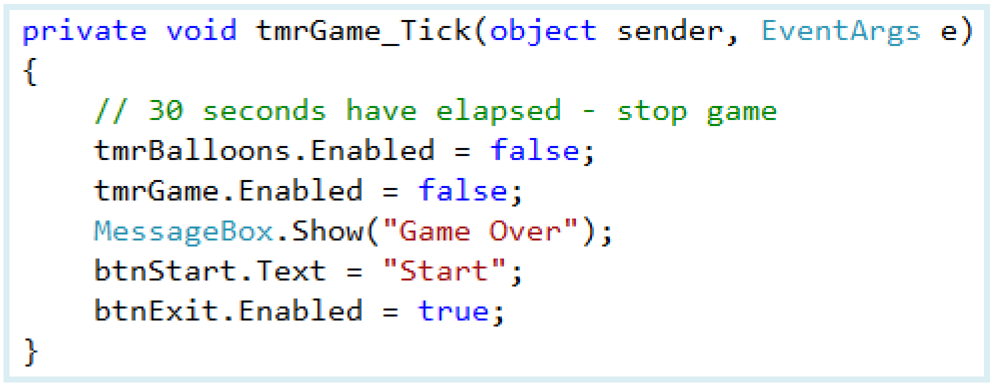
- Display 'Game Over' message
- Set btnStart Text to "Start"
- Enable btnExit button
- Stop the timers

Look at the **btnStart_Click** event method and see if you can identify all of the outlined steps. Notice the balloons are positioned just above the panel and the speeds are set using the formula given earlier:

```
private void btnStart_Click(object sender, EventArgs e)
{
    if (btnStart.Text == "Start")
    {
        // New Game
        myGraphics.Clear(pnlBalloon.BackColor);
        btnStart.Text = "Stop";
        btnExit.Enabled = false;
        score = 0;
        lblScore.Text = score.ToString();
        // set each balloon off top of panel and give new speed
        for (int i = 0; i < 5; i++)
        {
            balloonY[i] = -balloonSize;
            balloonSpeed[i] = myRandom.Next(3,7);
        }
        // Set arrow near center
        pinX = (int)(pnlBalloon.Width / 2);
        myGraphics.DrawImage(picPin.Image, pinX,
            pnlBalloon.Height - pinSize, pinSize, pinSize);
        // Give form focus so it can accept KeyDown events
        this.Focus();
        tmrBalloons.Enabled = !(tmrBalloons.Enabled);
        tmrGame.Enabled = !(tmrGame.Enabled);
    }
    else
    {   // Game stopped
        btnStart.Text = "Start";
        btnExit.Enabled = true;
        tmrBalloons.Enabled = !(tmrBalloons.Enabled);
        tmrGame.Enabled = !(tmrGame.Enabled);
        MessageBox.Show("Game Over");
    }
```

Save and run the project. Make sure you get no run-time errors. The pin should be centered on the panel. Make sure the pin motion keys (A and F) still work OK. Stop the project.

The **btnStart_Click** event method toggles the two timer controls. What goes on in the two Tick events? We'll do the easy one first. Each game lasts 30 seconds. This timing is handled by the tmrGame timer. It has an Interval of 30000, which means it's Tick event is executed every 30 seconds. We'll only execute that event once - when it is executed, we stop the game. The code to do this is identical to the code executed if the btnStart button is clicked when its Text is **Stop**. The **tmrGame_Tick** event method should be:



```
private void tmrGame_Tick(object sender, EventArgs e)
{
    // 30 seconds have elapsed - stop game
    tmrBalloons.Enabled = false;
    tmrGame.Enabled = false;
    MessageBox.Show("Game Over");
    btnStart.Text = "Start";
    btnExit.Enabled = true;
}
```

Save the project. Run it. Click **Start**. Play with the pin motion keys or just sit there. After 30 seconds, you should see the 'Game Over' notice pop up and see the buttons change appearance. If this happens, the tmrGame timer control is working properly. If it doesn't happen, you need to fix something. Stop the project.

Now, to the heart of the Balloonoons game - the **tmrBalloons_Tick** event. We haven't seen any dropping balloons yet. Here's where we do that, and more. The tmrBalloons timer control handles the animation sequence. It drops the balloons down the screen, checks for popping, and checks for balloons reaching the bottom of the panel. It gets new balloons started. There's a lot going on. The method steps are identical for each balloon. They are:

- Move the balloon.
- Check to see if balloon has popped. If so, sound a beep, make the balloon disappear, increment score and make balloon reappear at the top with a new speed.
- Check to see if balloon has reached the bottom without being popped. If so, start a new balloon with a new speed.

The steps are easy to write, just a little harder to code. Moving a balloon simply involves erasing it at its old location and redrawing it at its new location (determined by the balloonY value). To check if the balloon has reached the bottom, we use the border crossing logic discussed earlier. The trickiest step is checking if a balloon has popped. One way to check for a balloon pop is to check to see if the balloon image rectangle overlaps the pin rectangle using the collision detection. This would work, but would a balloon pop if the pin barely touched the balloon? In our code, we modify the collision logic such that we will not consider a balloon to be popped unless the <u>entire</u> width of the pin is within the width of the balloon.

Here's the complete **tmrBalloon_Tick** event implementing these steps. The balloons are handled individually within the structure of a **for loop**:

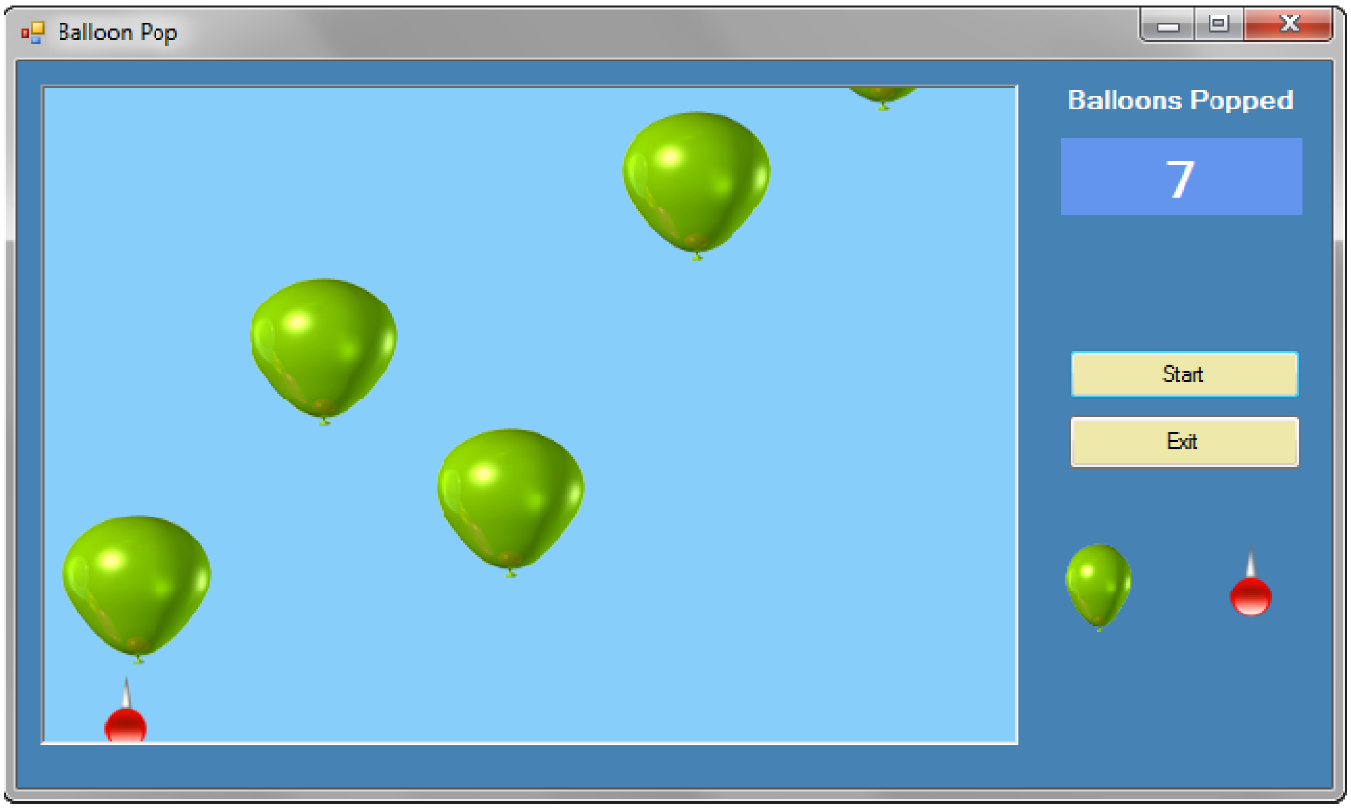
```
private void tmrBalloons_Tick(object sender, EventArgs e)
{
    for (int i = 0; i < 5; i++)
    {
        // erase balloon
        myGraphics.FillRectangle(blankBrush, balloonX[i], balloonY[i],
            balloonSize, balloonSize);
        // move balloon
        balloonY[i] = balloonY[i] + balloonSpeed[i];

        // check if balloon has popped
        if ((balloonY[i] + balloonSize) > (pnlBalloon.Height - pinSize))
        {
            if (balloonX[i] < pinX)
            {
                if ((balloonX[i] + balloonSize) > (pinX + pinSize))
                {
                    // Balloon has popped
                    // Increase score - move back to top
                    score += 1;
                    lblScore.Text = score.ToString();
                    balloonY[i] = -balloonSize;
                    balloonSpeed[i] = myRandom.Next(3, 7);
                }
            }
        }
        // check for moving off bottom
        if ((balloonY[i] + balloonSize) > pnlBalloon.Height)
        {
            // Balloon reaches bottom without popping
            // Move back to top with new speed
            balloonY[i] = -balloonSize;
            balloonSpeed[i] = myRandom.Next(3, 7);
        }
        // redraw balloon at new location
        myGraphics.DrawImage(picBalloon.Image, balloonX[i], balloonY[i],
            balloonSize, balloonSize);
    }
}
```

Do you see how all the steps are implemented?

**Run the Project**

Run the project. Make sure it works. Make sure each balloon falls. Make sure when a balloon reaches the bottom, a new one is initialized. Make sure you can pop each balloon. And, following a pop, make sure a new balloon appears. Make sure the score changes by one with each pop. Here's what my screen looks like in the middle of a game:



By building and testing the program in stages, you should now have a thoroughly tested, running version of Balloons. If you do find any bugs and need to make any changes, make sure you resave your project.

**Level 4+ Achievements**

- When a balloon pops, it just disappears from the screen. Maybe change the Image property of the picture box control. Or flash the panel background color.
- Add selectable difficulty levels to the game. This could be used to make the game easy for little kids and very hard for experts. What can you do to adjust the game difficulty? One thing you could do is adjust the size of the popping pin. To pop a balloon, the entire pin width must fit within the width of a balloon. Hence, a smaller pin would make it easier to pop balloons before they reach the bottom of the picture box. A larger (wider) pin makes popping harder. The balloon dropping speed also affects game difficulty. Slowly dropping balloons are easy to pop - fast ones are not
- Make it possible to play longer games and, as the game goes on, make the game more difficult using some of the ideas above (smaller pin, faster balloons). You've seen this in other games you may have played - games usually get harder as time goes on.
- Players like to know how much time they have left in a game. Add this capability to your game. Use a label to display the number of seconds remaining. You'll need another timer control with an Interval of 1000 (one second). Whenever this timer's Tick event is executed, another second has gone by. In this event, subtract 1 from the value displayed in the label. You should be comfortable making such a change to your project.
- Another thing players like to know is the highest score on a game. Add this capability. Declare a new variable to keep track of the highest score.