Chapter 6A

Visual C# Program: Simple Game

In this chapter, you will learn how to use the following Visual C# Application functions to World Class standards:

- **Opening Visual C# Editor**
- **Beginning a New Visual C# Project**
- **Laying Out a User Input Form in Visual C#**
- **Inserting a Label into a Form**
- **Adding a PictureBox in Visual C#**
- **Adding Comments in Visual C# to Communicate to Others**
- **Declaring Variables in a Program with the Int Statement**
- **Setting Variables in a Program**
- **Adding a Timer to the Program**
- **Programming for the Timer**
- **Running the Program**

Open the Visual C# Editor

In this lesson, we will write our first game. We will find that this kind of program requires constant input from the user and so there are new functions for us to learn. Some of the C# code will be similar to the syntax written in Chapter 5. In our first game, we will launch a ball from the upper left area of the window and we will try to keep the ball from passing the imaginary line just equal to the paddle and if it does, we will lose a life. We gain points by hitting the ball back up to the top of the window where it will ricochet and return to us on another route. The number of lives remaining will be posted on the top of the window and if we lose our last life, the game over text will appear and the game will conclude.

To open this new project, we select File on the Menu Bar and New Project.

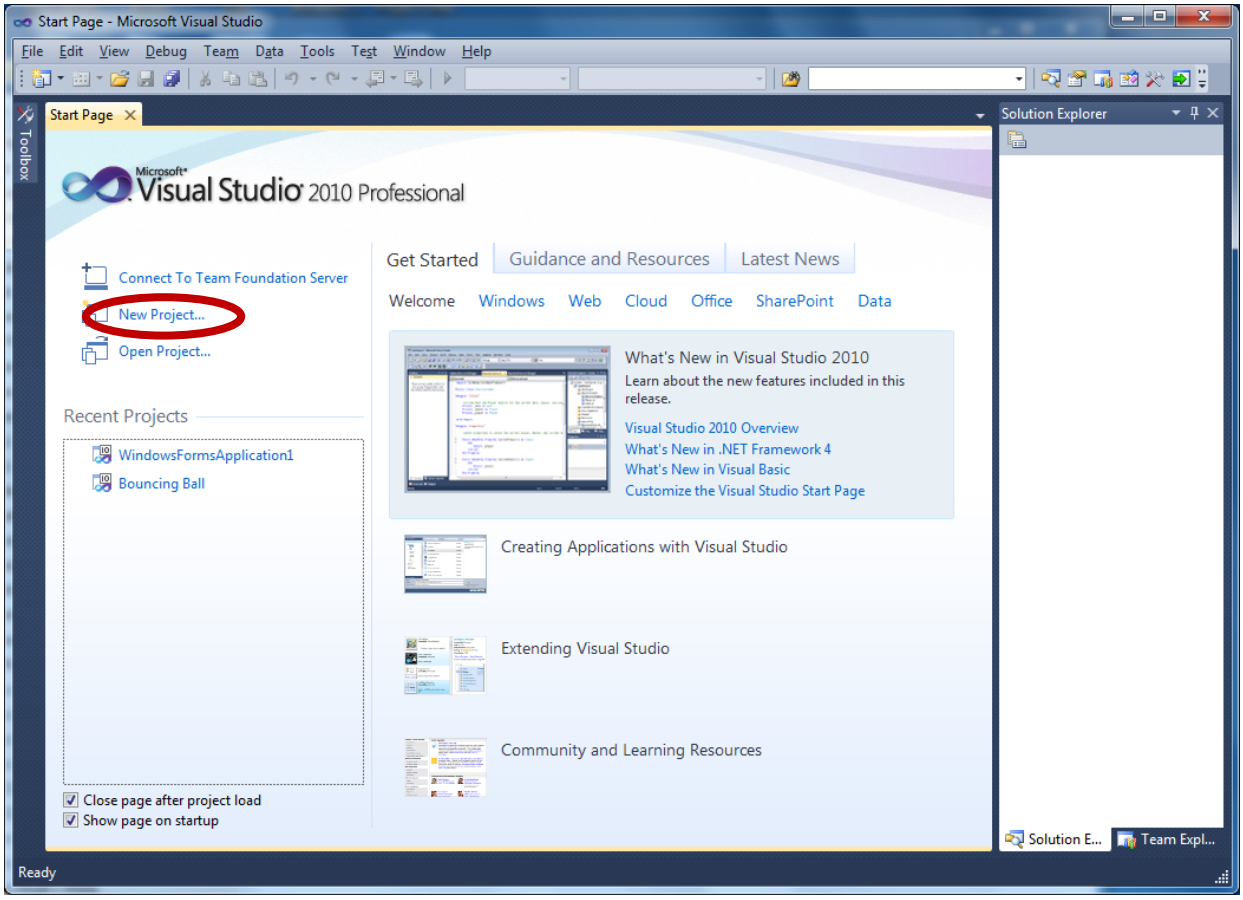

Figure 6A.1 – The Start Page

To open a new project, we select New Project on the left side of the Start Page.

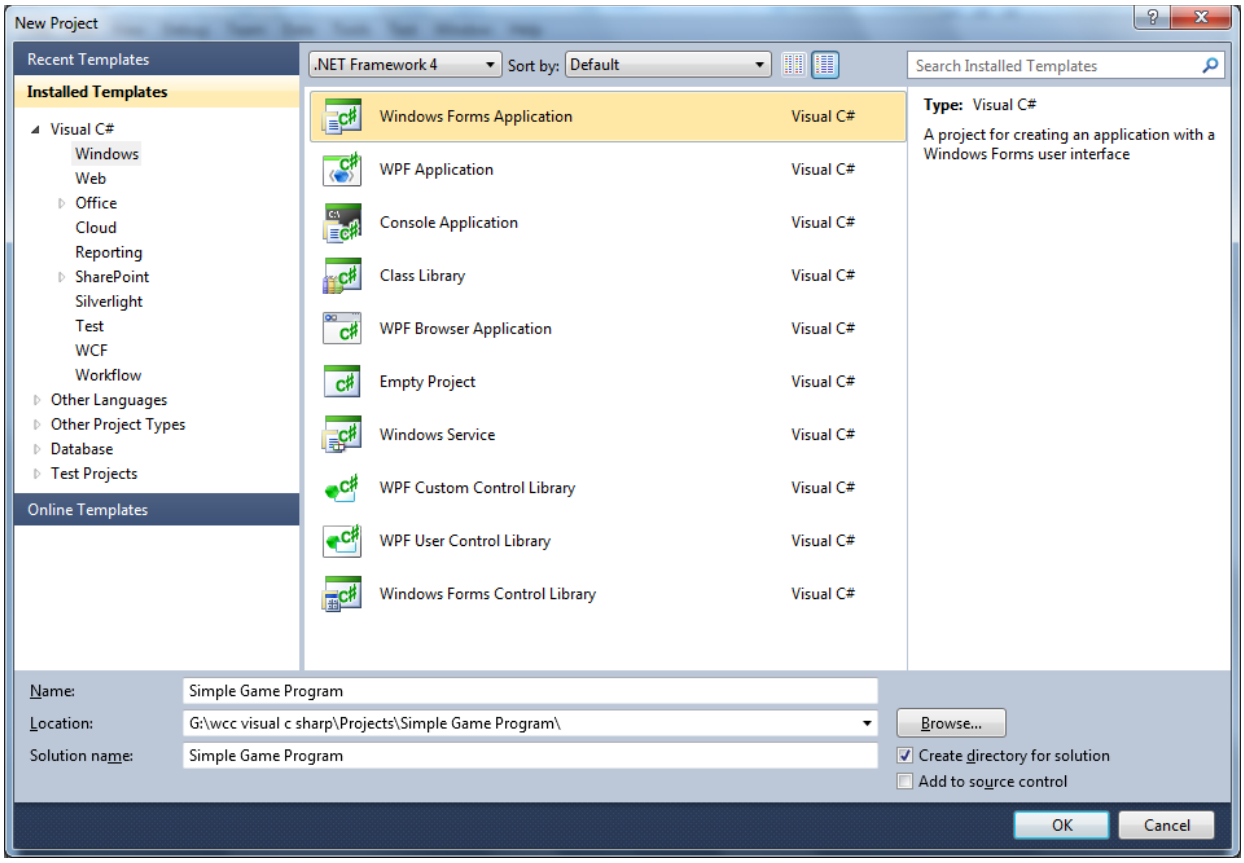

Figure 6A.2 – New Project

We start a new Windows Application Project by picking the Windows under Visual C # in the left pan of the New Project window. Then we pick Windows Form Application in the center pane.

At the bottom of the Window, we name the project, Simple Game Program. We make a folder for our projects called Visual C Sharp on the desktop, on our flash drive or in the Documents folder. We make another folder inside the first called Simple Game Program. On the New Project window, we browse to the Simple Game Program location. The solution name is the same as the project name.

Beginning a New Visual C# Application

Remember, that all programming projects begin with one or more sketches. The sketch will show labels and pictures. In this project, we will name the input, **Simple Game**. In this application, we will just have a form.

We will write code that moves a maroon ball from the upper left corner starting at the 1,1 location. We will displace the ball 1 pixel to the right and 1 pixel down for each tick of time on the timer. We will have a horizontal image for a paddle to stop the ball from going by us. We will add a label for three lives and another for keeping the score. We will gain a point for every time the ball is returned upward and the game will continue. The name of the game and the title of the score boxes are in three labels.

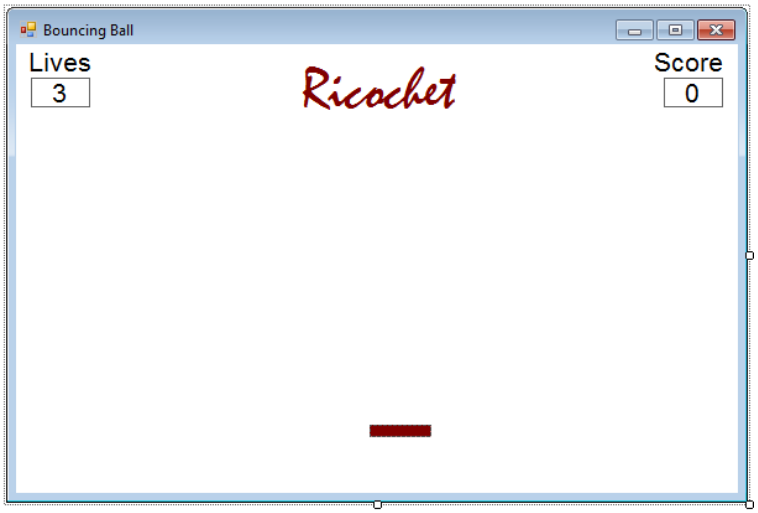

Figure 6A.3 – Sketch of the Simple Game Form

If the player loses all of their lives, we will have a label that says "Game Over" appear at the bottom of the window and the game will conclude.

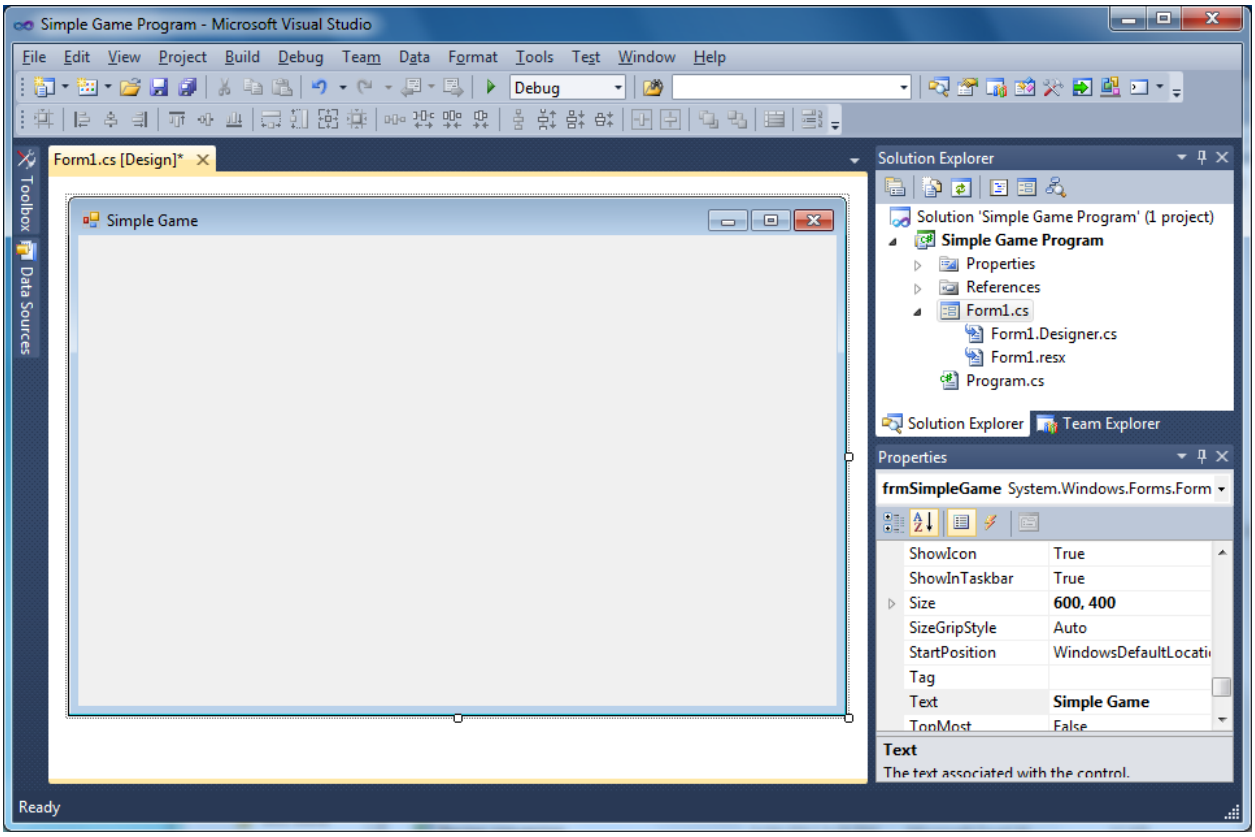

Figure 6A.4 – Designing the Simple Game Form in Visual C#

Laying Out a User Input Form in Visual C#

We will change the **Text** in the Properties pane to Simple Game to agree with the sketch in Figure 6A.3. Go ahead and change the form in two other aspects, BackColor and Size.

Alphabetic	
BackColor	White
Size	600, 400

The first number is the width and the second number is the height. The form will change in shape to the size measurement.

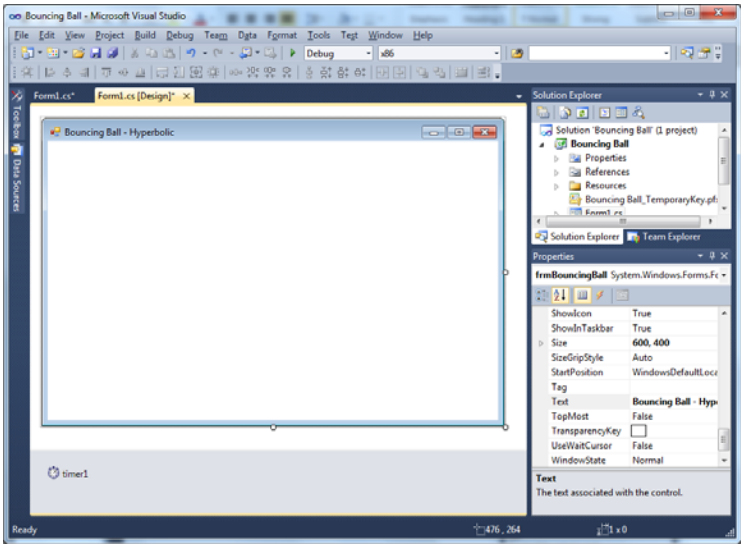

Figure 6A.5 – Setting the Form Properties

The background color will change to a white. There are many more attributes in the Properties pane that we will use on future projects.

In the Solution Explorer pane, right click on Form1.cs and rename it to frmSimpleGame.cs.

Inserting a Label into a Form

A good form is easy to figure out by the user, so when we are attempting to provide information on the window that will run in Windows; we add labels to textboxes to explain our intent. Press the Label (A) button on the Control Toolbar to add a label. To size the label area, click on the upper left area of the form and hold down on the left mouse button, draw the dotted label box.

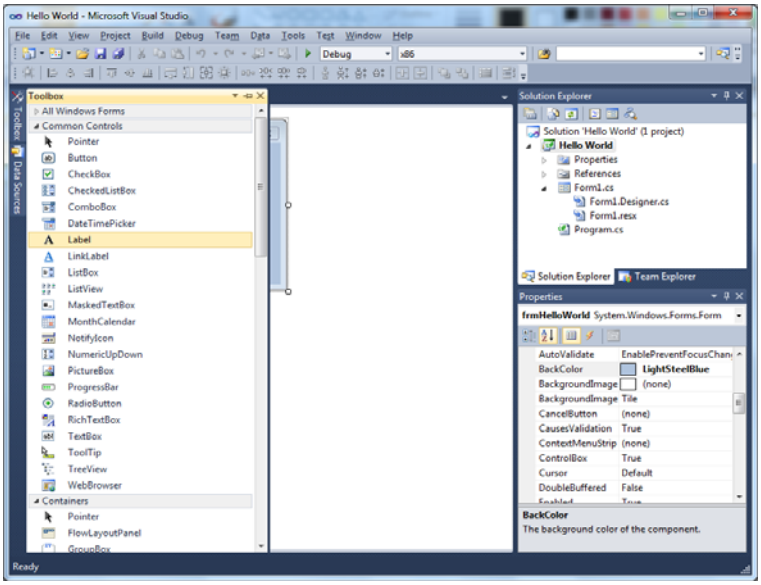

Figure 6A.6 – Placing a Label on the Form

We will name the Label using a common Visual C# naming convention where the programming object is a three letter prefix followed by the name or phrase of the tool. For our first label, the name is **lblLivesLabel.**

Alphabetic	
(Name)	lblLivesLabel
BackColor	White
Font	Arial, 16 pt
Text	Lives

On the sketch, the label's caption is "**Lives**" The font on the sketch is 16 point, Arial Narrow. When highlighting the row for Font, a small command button with three small dots appears to the right of the default font name of Microsoft San Serif. Click on the three dotted button to open the Visual C# Font window.

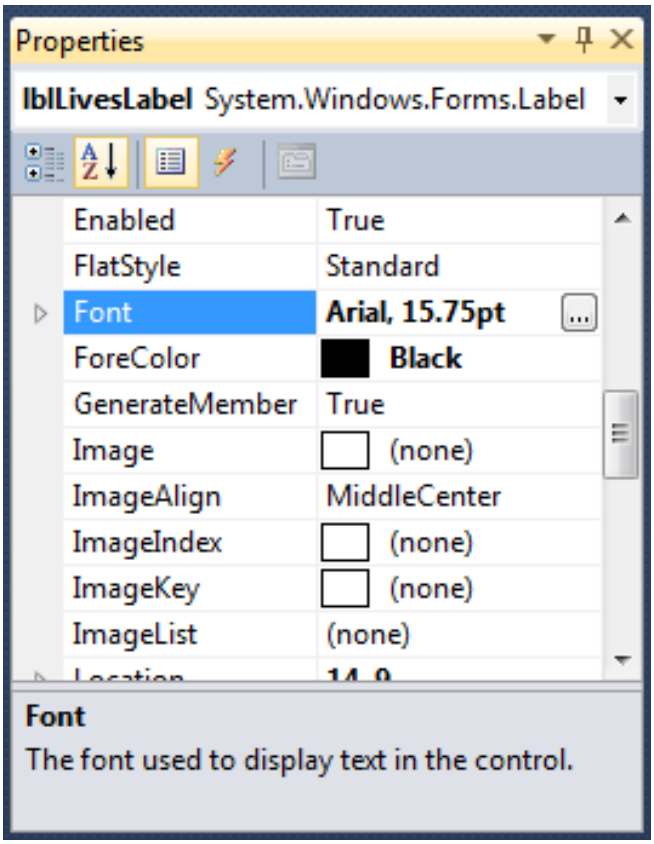

Figure 6A.7 – Changing the Font Property

We will select the Arial font, Regular font style and 16 size for this project to agree with the initial sketch if the user input form. When we adjust the attributes for the label, these changes do not alter globally for the other objects on the form. If we wish to underline the text or phrase in the label, add a check to the Underline checkbox in the Effects section of the Font window. When we finish making changes to the font property, select the OK command button to return to the work area.

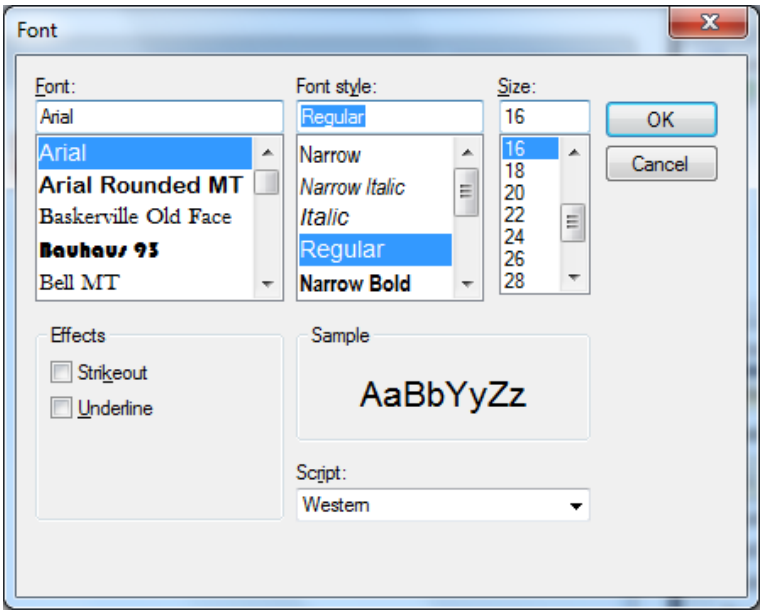

Figure 6A.8 – The Font Window in Visual C#

When the first label is done, the background color of the label matches the background color of the form. In many cases that effect is visually pleasing to the eye, versus introducing another color. Both color and shape will direct the user in completing the form along with the explanation we place on the window to guide the designer in using the automated programs. Use colors and shape strategically to communicate well.

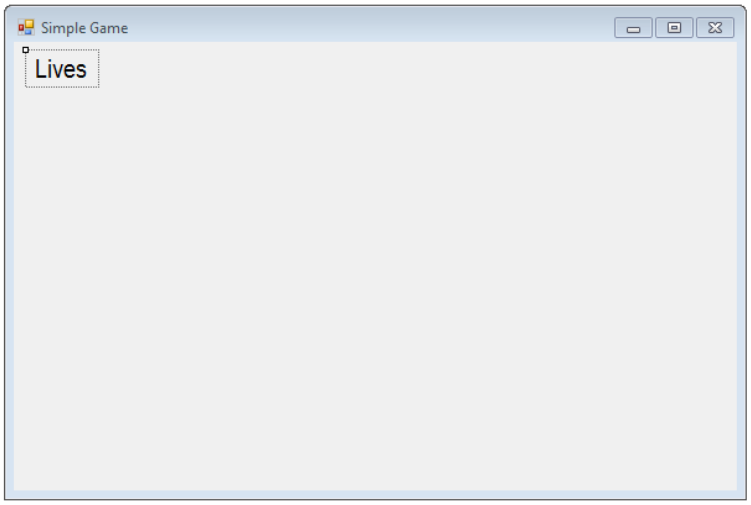

Figure 6A.9 – The Finished Label on the Form

We will call the label for the score lblScore and we will make the text state "score".

Alphabetic	
(Name)	lblScoreLabel
BackColor	White
Font	Arial, 16 pt
Text	Score

The next label is for the name Ricochet which is lblName.

Alphabetic	
(Name)	lblName
BackColor	White
Font	Mistral, 36 pt
ForeColor	Maroon
Text	Ricochet

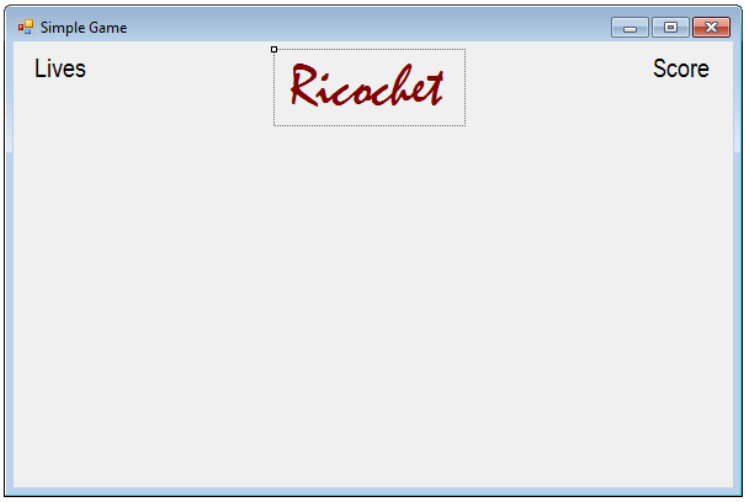

Figure 6A.10 – The Finished Labels on the Form

We will add two labels that will hold the number of lives and our score. We will position these labels under their identifiers and they will hold their initial number, 3 for lives and 0 for our score. We will place the last label in a position below the paddle and in the center.

Alphabetic	
(Name)	lblScore
Border	Fixed Single
BackColor	White
Font	Arial, 16 pt
Text	0

Alphabetic	
(Name)	lblLife
Border	Fixed Single
BackColor	White
Font	Arial, 16 pt
Text	3

Alphabetic	
(Name)	lblGameOver
BackColor	White
Font	Arial, 16 pt
Text	(blank)

Figure 6A.11 – Placing a Labels on the Form

After typing in Game Over for the last label, we clear out the text so we will insert the text in the object when the game is concluded.

Inserting a Picture into the Form

We will place two pictures on the form, one is the paddle and the other is the ball. The paddle is 50 pixels wide by 10 pixels tall and the ball is 10 by 10 pixels. We need to open a graphics program and make both of these icons and save them in our Simple Game Program project folder.

We select the toolbox and Picturebox and we draw a box to the right of the labels that will contain the answers. We name the picturebox **imgPaddle**. We scroll down on the properties window and select the three dots button at the Image property and a Select Resource window will appear.

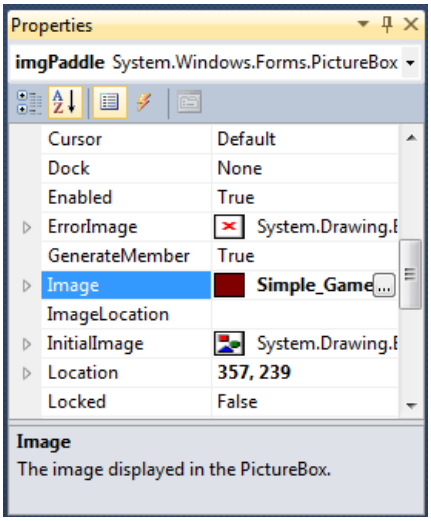

Figure 6A.12 – Adding an Image

We then will import the graphic of our paddle which we made in Microsoft Paint and saved as a bitmap image. We then press the OK button and the image will appear in the picturebox.

Alphabetic	
(Name)	imgPaddle
Image	Paddle.bmp
Location	260,320
Size	50,10

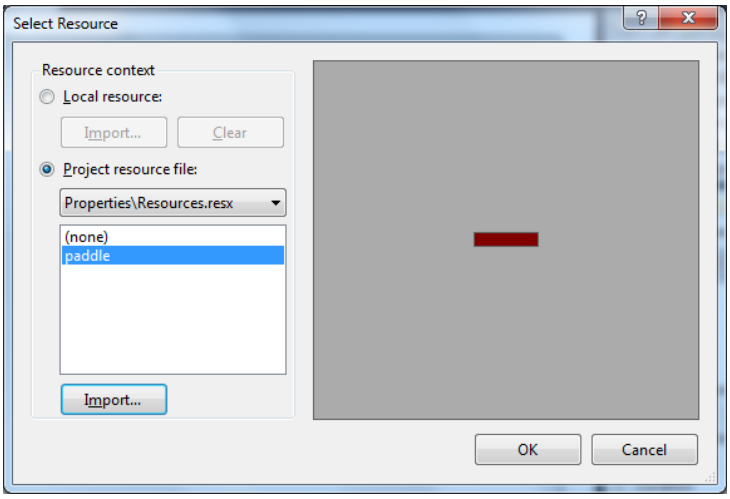

Figure 6A.13 – Import the Paddle Image

We then will import the graphic of our ball which we made in Microsoft Paint and saved as a JPEG image. We then press the OK button and the image will appear in the picturebox.

Alphabetic	
(Name)	imgBall
Image	Ball.jpg
Location	1,1
Size	10,10

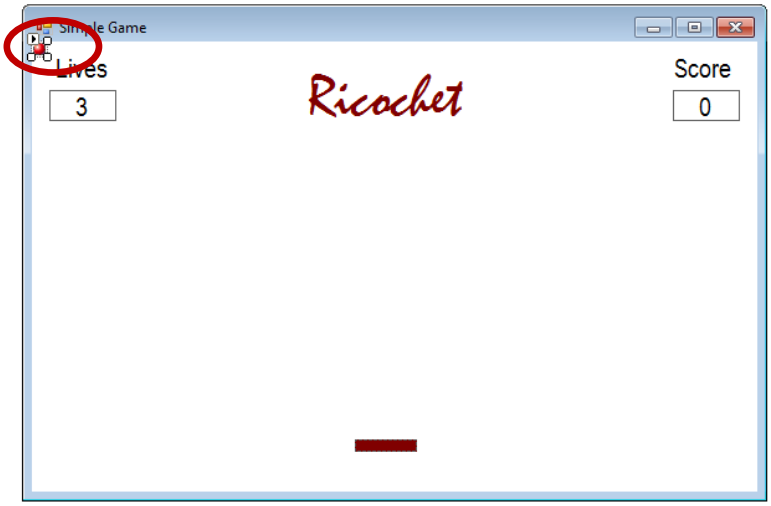

Figure 6A.14 – Import the Ball Image

Adding Comments in Visual C# to Communicate to Others

The comments we placed in the first three lines of the program will inform the individual opening and reading the code of the ownership. This is for those user that may run the application without checking the label on the bottom of the form with the copyright information. It is a great tool to alert the client to the rules of the program and tell them what the application will do.

To begin the actual coding of the program, double click on the form. At the top of the program and after the line of code with Namespace Simple Game {, place the following comments with two slash (//) characters. Remember, the two slashes (//) will precede a comment and when the code is compiled, comments are ignored.

Type the following line of code:

```
///Simple Game
///This program will allow the player to return a ball and continue to earn points
///If the ball passes the paddle the player loses a point up to three times
```

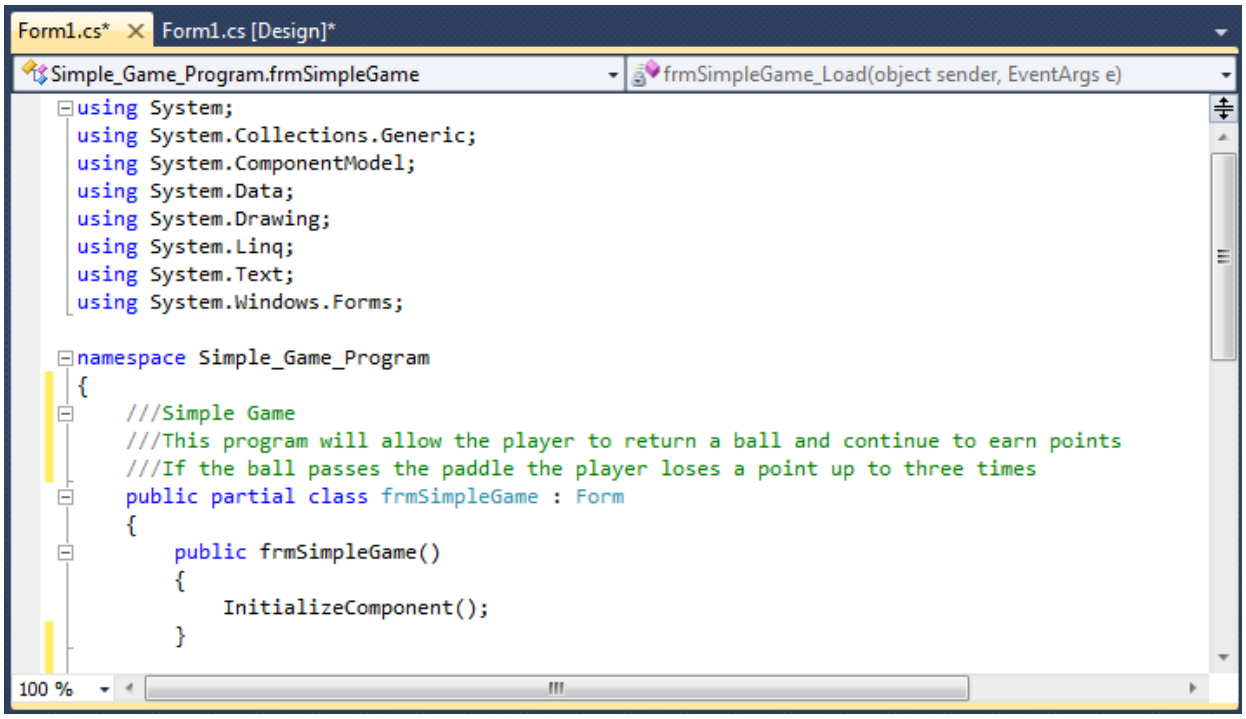

Figure 6A.15 – Adding an Introduction

Declaring Variables in a Program with the Int Statement

When we are going to use a number, text string or object that may change throughout the life of the code, we create a variable to hold the value of that changing entity. In Visual C#, the integer statement is one of the ways to declare a variable at the procedure level.

In this program, we will create data from mathematical computations. We will place the values in integer variables called x, y, dx (change of x), dy (change of y), ptx, pty for the paddle location, score for number of times we return the ball and lastly life for the number lives we have left. These variables will hold whole numbers for movement on the form in terms of pixels, so we will declare all eight as integers.

Type the following code under the public partial class frmBouncingBall : Form subroutine of the program.

```
public partial class frmSimpleGame : Form
{
    int x;          ///x is the position on the x axis from the upper left corner
    int y;          ///y is the position on the y axis from the upper left corner
    int dx;         ///speed is the change of x
    int dy;         ///speed is the change of y
    int ptx;        ///x coordinate of the paddle
    int pty;        ///y coordinate of the paddle
    int score;      ///how many time the ball hits the paddle
    int life;       ///number of lives starting at 3
```

The integers x and y represent the actual position of the ball in the Simple Game Program. The dx and dy are the change in the x position for movement of the timer. The ptx and pty are the location of the paddle so we can calculate the where the 50 pixel long line is to determine whether the ball hit that region. The score is originally set to zero and increases by one each time the ball is hit, so we need a variable for it. The life integer starts at three and decreases by one each time the ball passes the paddle. When the integer life reaches zero, we will end the game.

Therefore, we can see the purpose of each variable we choose in the program.

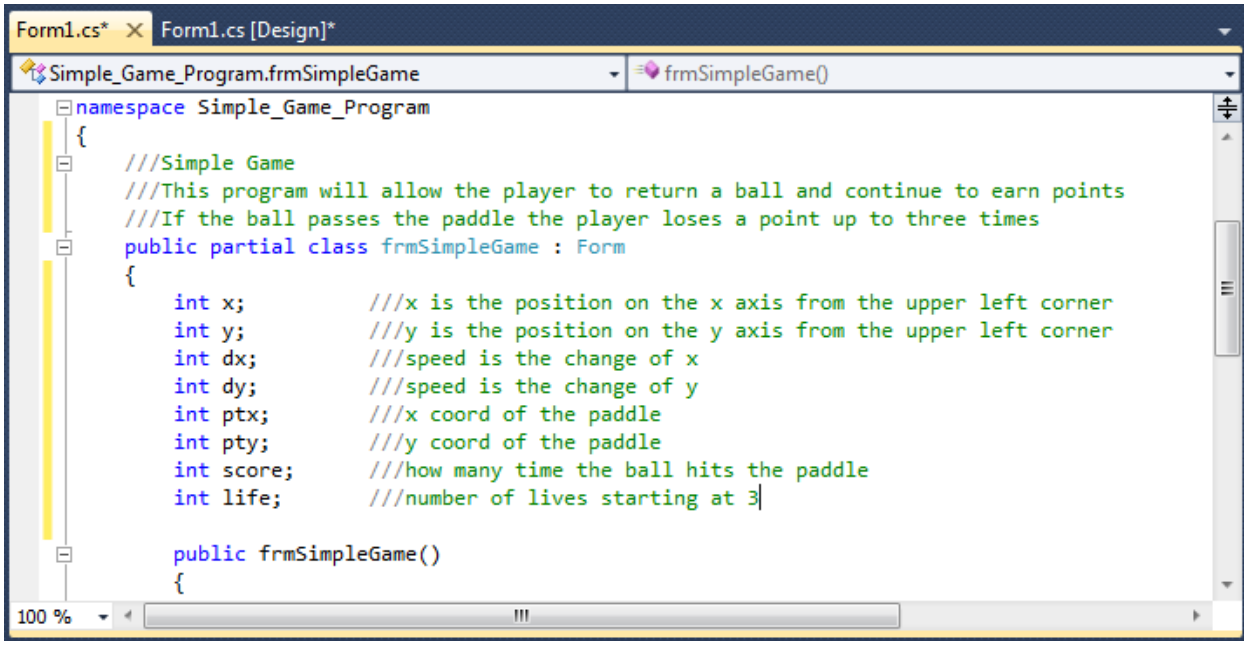

Figure 6A.16 – Declaring Variables with Int Statements

The variable names should relate to what data they hold. We should not have spaces in the name. Some programmers use the underscore character (_) to separate words in phrases. This is acceptable, but a double underscore (__) can cause errors if we do not detect the repeated character. We could call the variables x_coordinate and y_coordinate.

Setting Variables in a Program

Next, we will set the variables using the equal function. We will set the numbers in the two textboxes to their variable and we compute resistance by division and the power by using multiplication.

Type this code under the private void frmSimpleGame_Load(object sender, EventArgs e) subroutine of the program.

```
private void frmSimpleGame_Load(object sender, EventArgs e)
{
   //Loads the ball on the screen at the upper left corner of the window
   x = 1;       //x coordinate of the ball starting point
   y = 1;       //y coordinate of the ball starting point
   dx = 1;      //Speed of the ball in the x direction is 1
   dy = 1;      //Speed of the ball in the y direction is 1
   score = 0;  //The initial score is 0
   life = 3;    //The initial life is 3
}
```

The variable called speed is the change of vertical position in pixels, so we will start the movement of the ball in one pixel increments. The x and y coordinates are the position of the ball. The form is measured from 0,0 in the upper left corner to width and height of the form in the lower right corner, which is 600 pixels by 400 pixels for our project.

We start the ball at x equals 1 and y equals 1. We keep the change of x (dx) and the change of y (dy) the same throughout the entire program at 1. The score starts out at 0 and our life counter starts at 3.

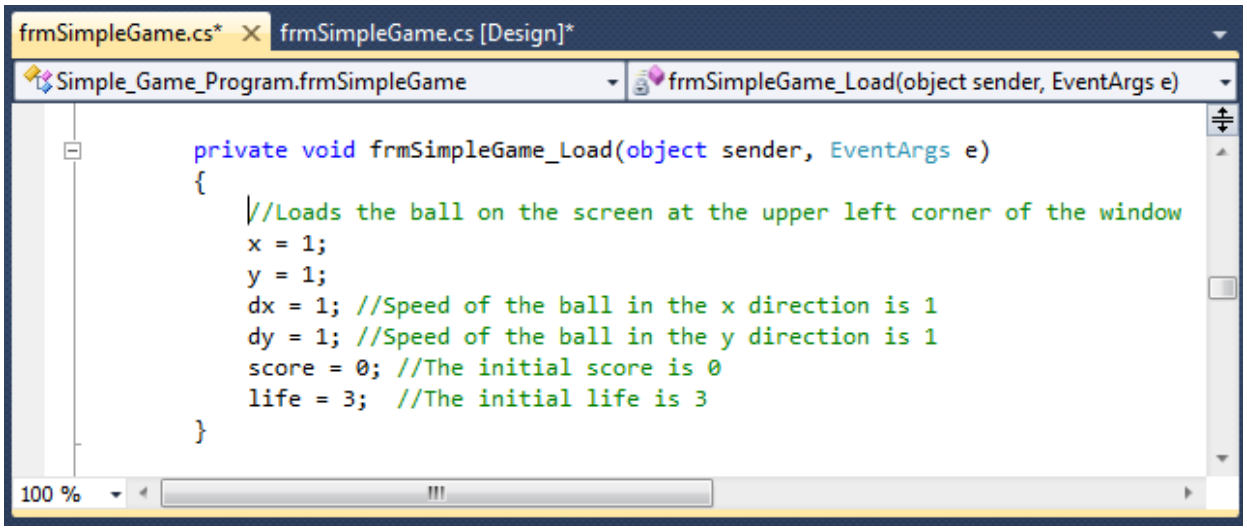

Figure 6A.17 – Setting the Variables in the C# Code

Adding a Timer to the Program

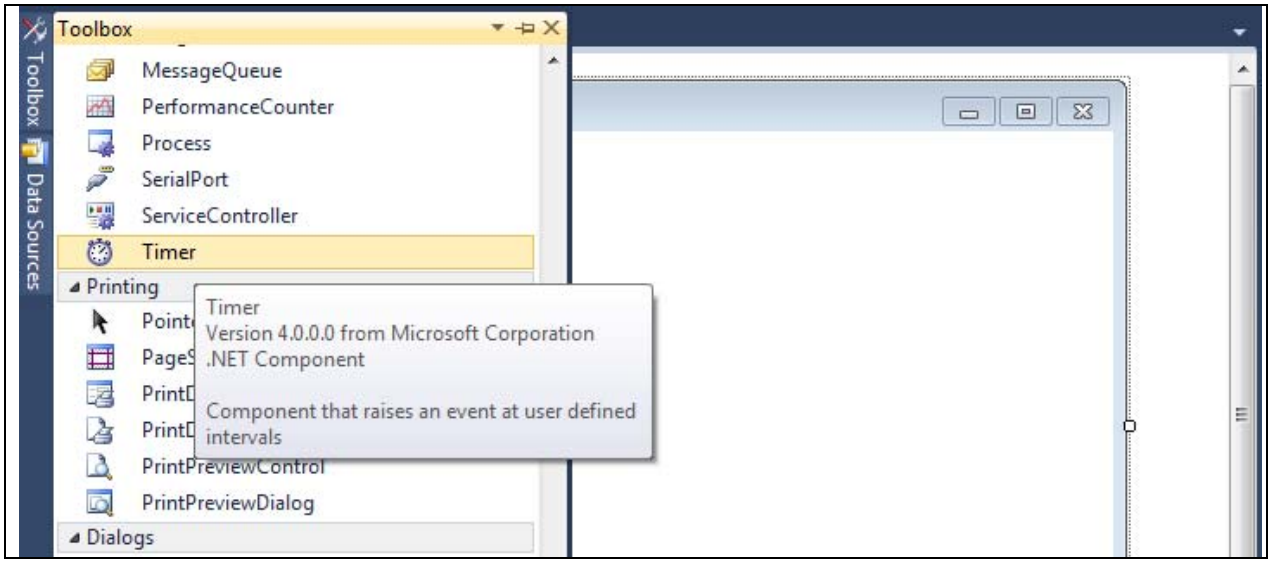

Figure 6A.18 – Adding a Timer

We will next add a Timer to the Simple Game application by selecting Timer from the Toolbox and placing it on the form.

We Double click on the timer and name the object timer1. We will set Enabled and GenerateMember as True and the Interval to 1 millisecond. We will set the Modifiers to Private.

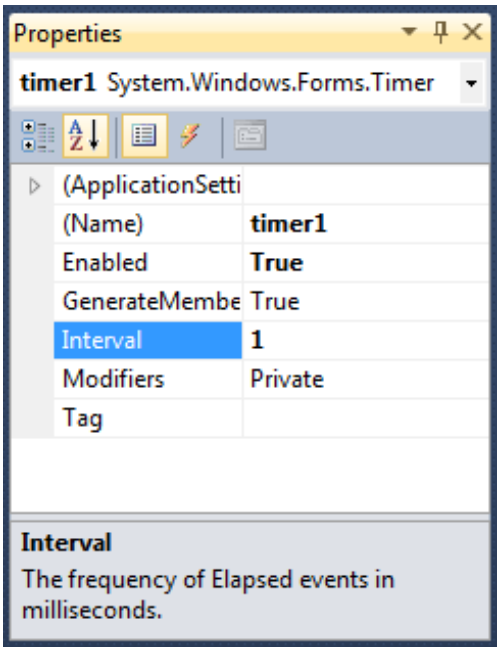

Figure 6A.21 – Setting the Timer's Properties

Programming for the Timer

We will double click on the timer at the bottom of the form and we add code to change the x and y coordinates, the direction of the ball using dx and dy, We also will add points to the score and deduct points from the number of lives left.

The first thing we do in the timer subroutine is set the ball location. Then we change the x coordinate by the value of dx. If the ball is at either the left vertical boundary (x < 0) or the right vertical boundary (x + 10 > this.clientsize.width), then we change the dx direction. This.ClientSize.Width is the number of pixels of the Window's width.

The next order of coding is to add dy to the y coordinate. If the ball is at the top horizontal boundary (y < 0), then we change the dy direction. The next check is to examine three points of logic. First we retrieve the location of the paddle. With an If statement, we will see if the ball's x coordinate is equal or greater than the paddle's location and less or equal than the width of the paddle. Finally, we check if the y + 10 of the ball is greater than the y coordinate of the paddle and if all conditions are met, we change the direction of the ball, add one to the score and post the new score on the game window. The next If statement will detect that the ball has gone by the paddle. Then we have an If statement to check to see if the game is over.

Type the following code in the private void timer1_Tick(object sender, EventArgs e) subroutine.

```
private void timer1_Tick(object sender, EventArgs e)
   {
   imgBall.Location = new Point(x, y);
   x = x + dx;
   if (x < 0)
   {
      dx = -dx; /// if y is less than 0 then we change direction
   }
   else if (x + 10 > this.ClientSize.Width)
   {
      dx = -dx; /// if y + 10, the radius of the circle is greater than the form height then we change direction
   }

   y = y + dy;
   if (y < 0)
   {
      dy = -dy; /// if y is less than 0 then we change direction
   }

   /// checks if the ball hits the paddle
   Point point1 = this.imgPaddle.Location;
   ptx = (point1.X);
   pty = (point1.Y);
   if (x >= ptx)
```

```
{
    if (x <= (ptx + 50))
    {
        if ((y + 10) > pty)
        {
            dy = -dy;
            score = score + 1;
            lblScore.Text = Convert.ToString(score);
        }
    }
}

/// checks if the ball goes by the paddle imaginary line
if ((y + 9) > pty)
{
    life = life - 1;
    lblLife.Text = Convert.ToString(life);
    dx = 1;
    dy = 1;
    x = 1;
    y = 1;
}

/// checks if the game is over
if (life == 0)
{
    lblGameOver.Text = "Game Over";
    x = -10;
    y = -10;
}

/// refresh the window
this.Invalidate();
}
```

Programming the Paddle

Brand new in this lesson is making left and right arrow keys move the paddle to the left and right in the game. We need to add the following to the public frmSimpleGame() portion of the code and right below InitializeComponent();.

```
KeyDown += new KeyEventHandler(frmSimpleGame_KeyDown);
```

Then we add the private void frmSimpleGame_KeyDown(object sender, KeyEventArgs e)

subroutine. The first two lines of code obtain the location of the paddle and assign it to px and py. Next if the right arrow is pressed, the paddle moves 5 pixels to the right. Next if the left arrow is pressed, the paddle moves 5 pixels to the left. The last if statement keeps the paddle in the window boundary. So if the key is pressed when the paddle is on the boundary, the location stays the same.

Type the following code:

```
public frmSimpleGame()
    {
        InitializeComponent();
        KeyDown += new KeyEventHandler(frmSimpleGame_KeyDown);
    }

private void frmSimpleGame_KeyDown(object sender, KeyEventArgs e)
    {
        int px = imgPaddle.Location.X;
        int py = imgPaddle.Location.Y;

        if (e.KeyCode == Keys.Right) px += 5;
        else if (e.KeyCode == Keys.Left) px -= 5;

        if (px < 0)
        {
            px = 0;
        }
        if (px > this.ClientSize.Width - 50)
        {
            px = this.ClientSize.Width - 50;
        }
        imgPaddle.Location = new Point(px, py);
    }
```

```
Form1.cs*  Form1.cs [Design]*
Simple_Game_Program.frmSimpleGame        frmSimpleGame_Load(object sender, EventArgs e)

        public frmSimpleGame()
        {
            InitializeComponent();
            KeyDown += new KeyEventHandler(Form1_KeyDown);
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            int px = imgPaddle.Location.X;
            int py = imgPaddle.Location.Y;

            if (e.KeyCode == Keys.Right) px += 5;
            else if (e.KeyCode == Keys.Left) px -= 5;

            if (px < 0)
            {
                px = 0;
            }
            if (px > this.ClientSize.Width - 50)
            {
                px = this.ClientSize.Width - 50;
            }
            imgPaddle.Location = new Point(px, py);
        }

100 %
```

Figure 6A.19 – Setting the Paddle Movement Properties

Running the Program

After noting that the program is saved, press the F5 to run the Simple Game application. The Simple Game window will appear on the graphical display. The ball will be moving down from the top and we have the ability to use the arrow keys to stop the ball from getting by the paddle.

After playing the Simple Game program, click on the red X in the upper right corner to close the application.

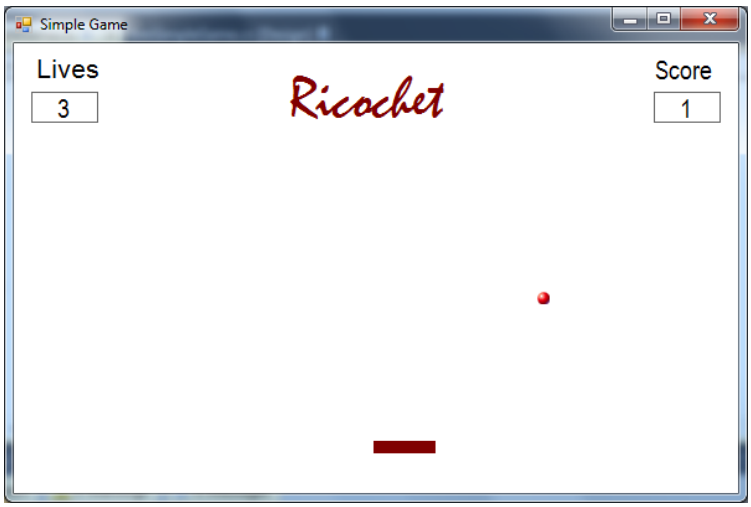

Figure 6A.20 – Launching the Program

If our program does not function correctly, go back to the code and check the syntax against the program shown in previous sections. Repeat any processes to check or Beta test the program. When the program is working perfectly, save and close the project.

Here is the entire code to check your syntax.

```
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 Simple_Game_Program
{
   //Simple Game
   //This program will allow the player to return a ball and continue to earn points
   //If the ball passes the paddle the player loses a point up to three times
   public partial class frmSimpleGame : Form
   {
      int x;         //x is the position on the x axis from the upper left corner
      int y;         //y is the position on the y axis from the upper left corner
      int dx;        //speed is the change of x
      int dy;        //speed is the change of y
      int ptx;       //x coord of the paddle
      int pty;       //y coord of the paddle
      int score;    //how many time the ball hits the paddle
      int life;       //number of lives starting at 3

      public frmSimpleGame()
      {
         InitializeComponent();
         KeyDown += new KeyEventHandler(frmSimpleGame_KeyDown);
      }

      private void frmSimpleGame_KeyDown(object sender, KeyEventArgs e)
      {
         int px = imgPaddle.Location.X;
         int py = imgPaddle.Location.Y;

         if (e.KeyCode == Keys.Right) px += 5;
         else if (e.KeyCode == Keys.Left) px -= 5;

         if (px < 0)
         {
```

```
            px = 0;
        }
        if (px > this.ClientSize.Width - 50)
        {
            px = this.ClientSize.Width - 50;
        }
        imgPaddle.Location = new Point(px, py);
    }

    private void frmSimpleGame_Load(object sender, EventArgs e)
    {
        //Loads the ball on the screen at the upper left corner of the window
        x = 1;
        y = 1;
        dx = 1; //Speed of the ball in the x direction is 1
        dy = 1; //Speed of the ball in the y direction is 1
        score = 0; //The initial score is 0
        life = 3;  //The initial life is 3
    }

    private void timer1_Tick(object sender, EventArgs e)
        {
        imgBall.Location = new Point(x, y);
        x = x + dx;
        if (x < 0)
        {
            dx = -dx; /// if y is less than 0 then we change direction
        }
        else if (x + 10 > this.ClientSize.Width)
        {
            dx = -dx; /// if y + 10, the radius of the circle is greater than the form height then we change direction
        }

        y = y + dy;
        if (y < 0)
        {
            dy = -dy; /// if y is less than 0 then we change direction
        }
        Point point1 = this.imgPaddle.Location;
        ptx = (point1.X);
        pty = (point1.Y);
        if (x >= ptx)
        {
            if (x <= (ptx + 50))
            {
                if ((y + 10) > pty)
                {
```

```
                dy = -dy;
                score = score + 1;
                lblScore.Text = Convert.ToString(score);
            }
        }
    }
    if ((y + 9) > pty)
    {
        life = life - 1;
        lblLife.Text = Convert.ToString(life);
        dx = 1;
        dy = 1;
        x = 1;
        y = 1;
    }
    if (life == 0)
    {
        lblGameOver.Text = "Game Over";
        x = -10;
        y = -10;
    }

    this.Invalidate();
    }
  }
}
```

There are many variations of this Visual C# Application we can practice and obtain information from a personal computer. While we are practicing with forms, we can learn how to use variables, strings and comments. These are skills that we want to commit to memory.

*** World Class CAD Challenge 90-9 * - Write a Visual C# Application that displays a single form and when executed, the program will show a Simple Game from the computer using mathematical computations.**

Continue this drill four times using some other form designs, each time completing the Visual C# Project in less than 1 hour to maintain your World Class ranking.