Chapter 6C

Visual C# Program: Simple Game 3

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 third simple game, which we saw Ruben Correa working on in the computer lab. The game uses the paddle from our first application, but now we will deflect the ball to hit the blocks that are arrayed across the top of the window. In our game, we will create a matrix of eight rectangles running horizontally across the dialogue box, but Ruben has added the boxes to 5 or 6 deep in column. His rendition of this lesson runs faster since he has set the change in x and y coordinates at 2 or 3 along with the rate of change when moving the paddle. As in all programs, we expect different code writers to use their imagination to create unique tools for their users. In this application, we intend the users to have fun.

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

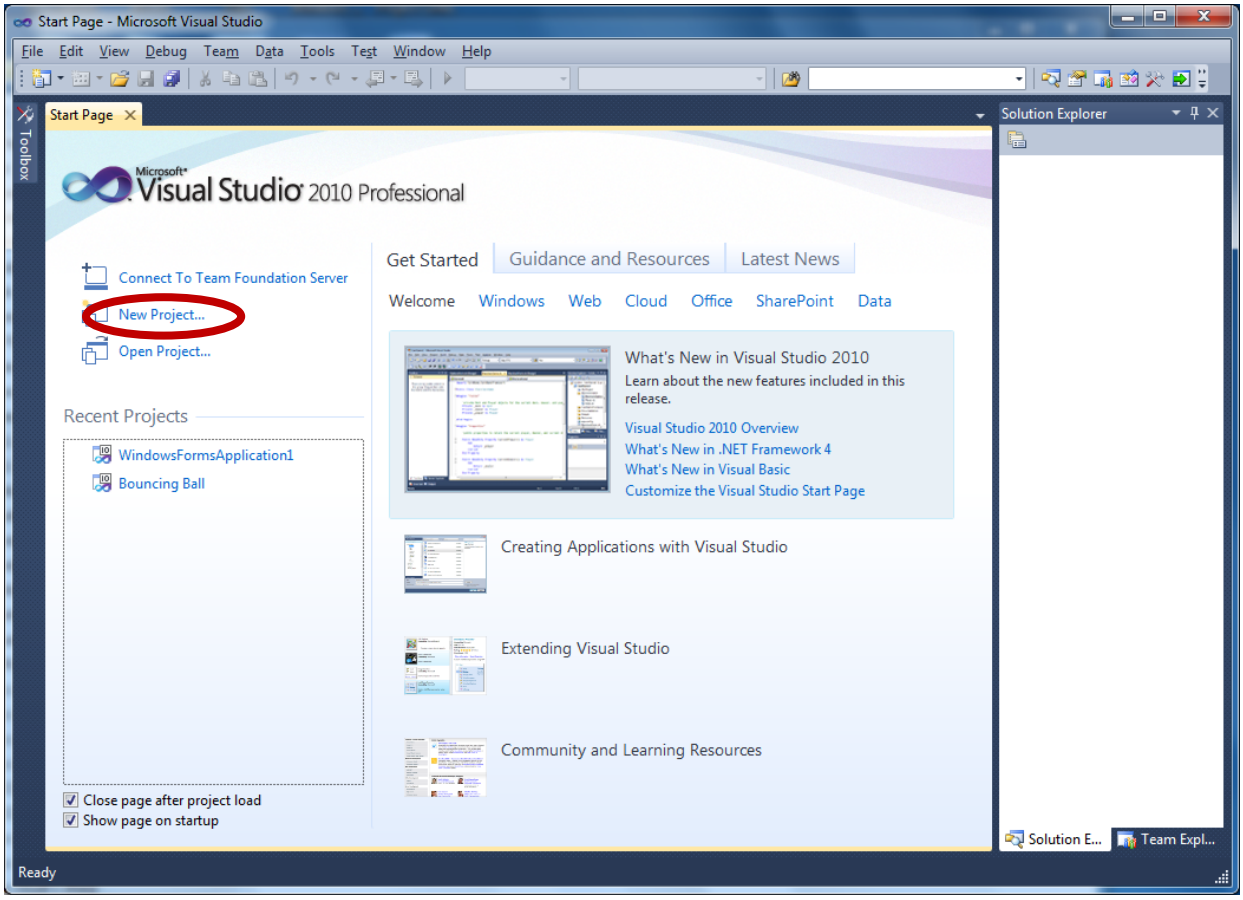

Figure 6C.1 – The Start Page

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

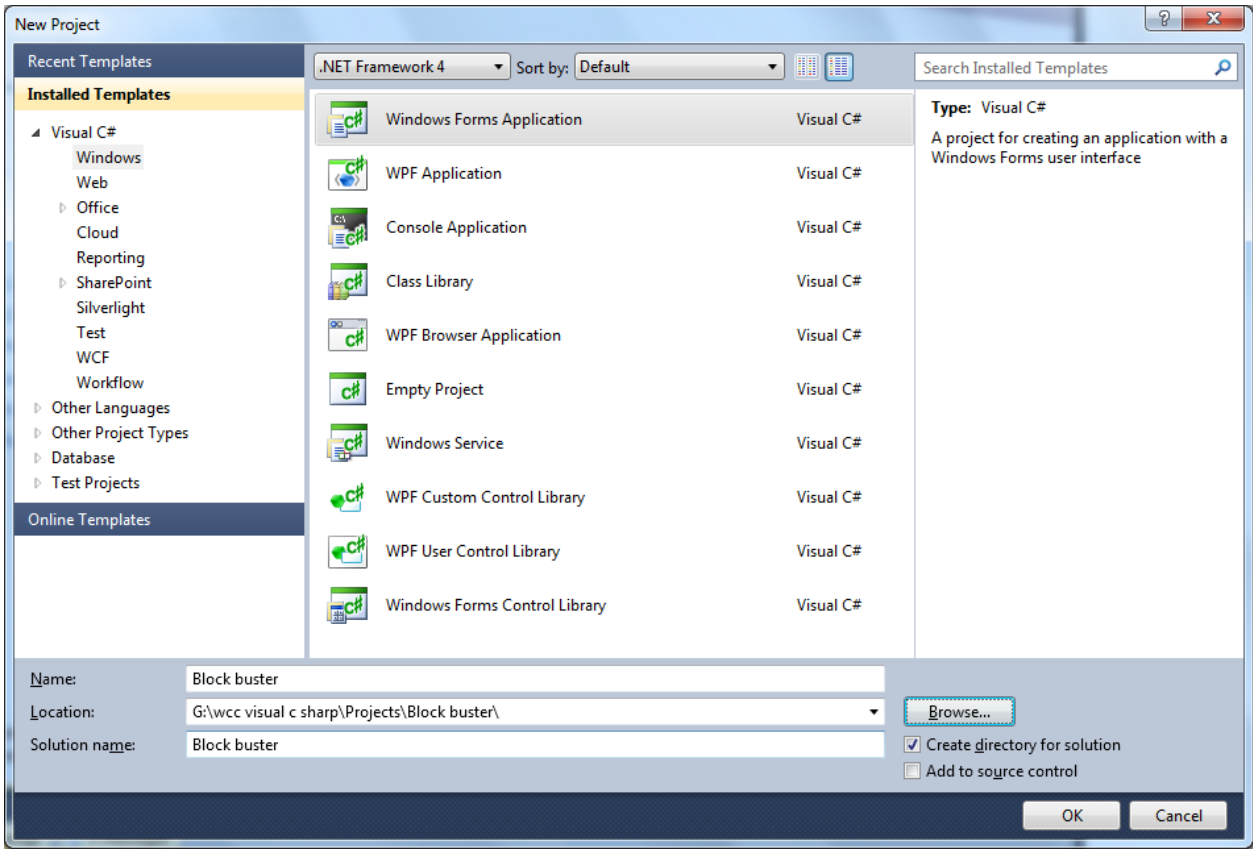

Figure 6C.2 – New Project

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, Block buster. 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 Block buster. On the New Project window, we browse to the Block buster 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 middle bottom starting at the 430,465 location. We will displace the ball 1 pixel to the left and 1 pixel up for each tick of time on the timer. We will align 90 by 20 pixel blocks across the top. We will add a label for keeping the score. We will gain a point for every time the ball hits our paddle and every instance a block is hit. The name of the game and the title of the score boxes are in three labels.

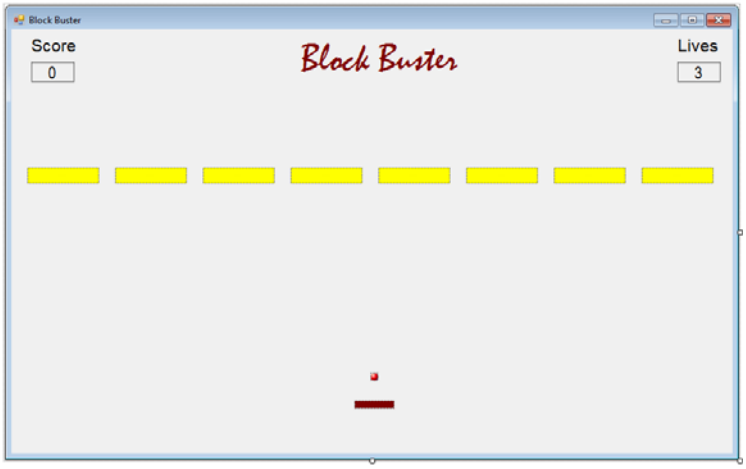

Figure 6C.3 – Sketch of the Simple Game Form

When all of the blocks are gone, we will have to continue to play, so the ball does not get by the paddle. If we lose our three lives, the game will conclude.

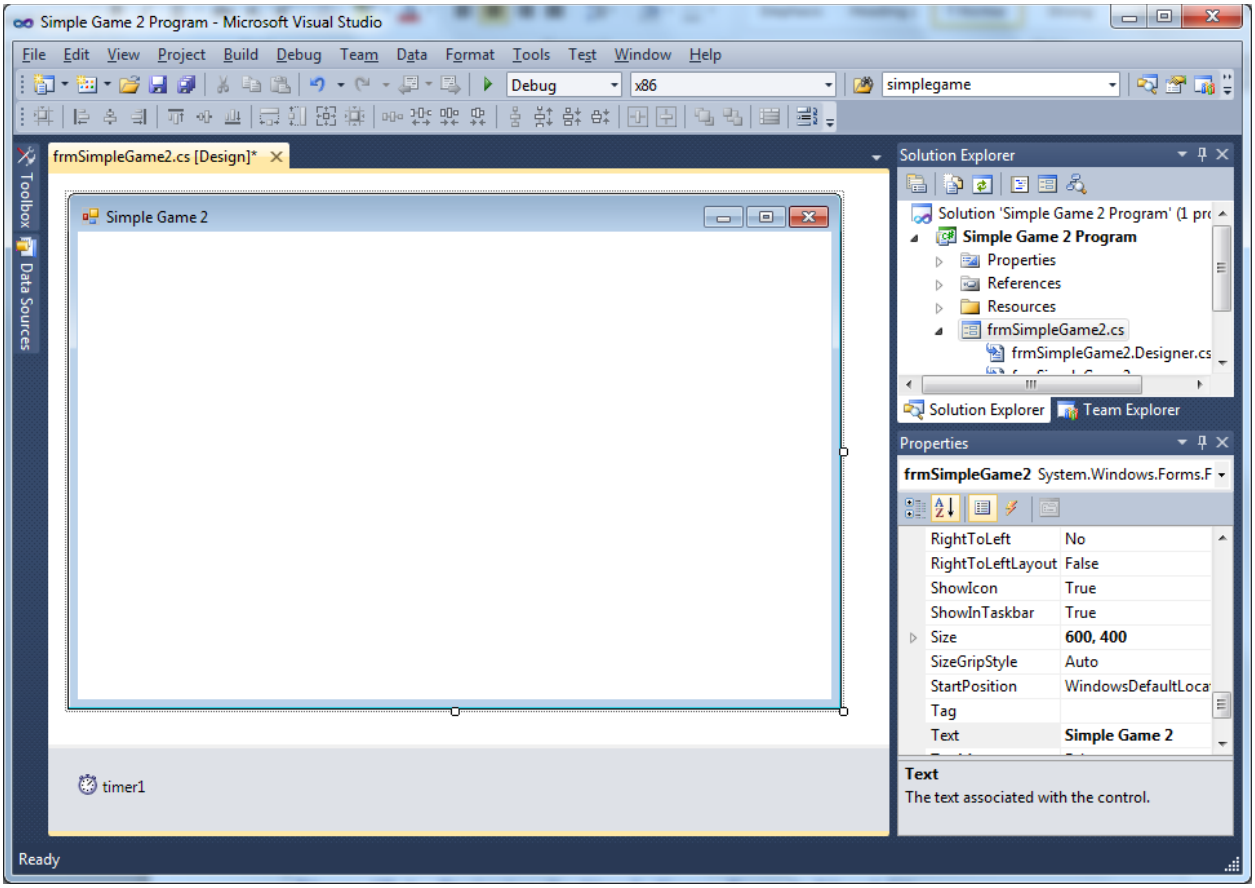

Figure 6C.4 – Designing the Simple Game Form in Visual C#

Laying Out a User Input Form in Visual C#

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

Alphabetic	
BackColor	White
Size	920, 569

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

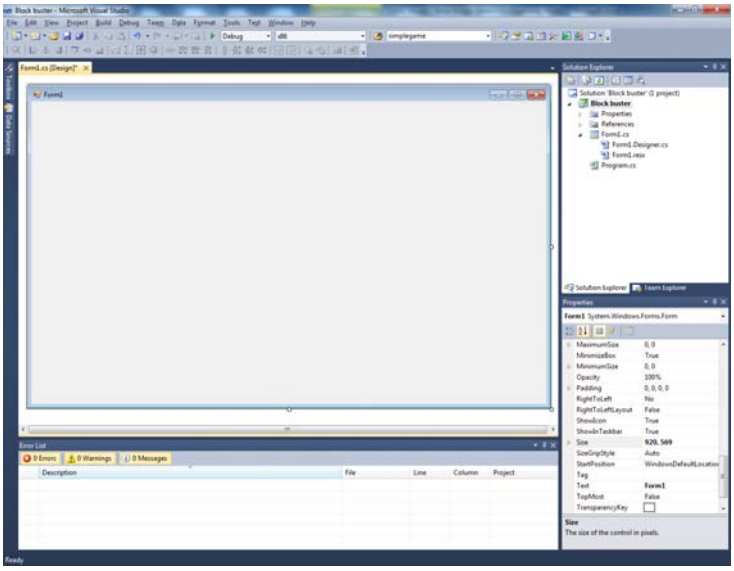

Figure 6C.5 – Setting the Form Properties

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 frmBlock_buster.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.

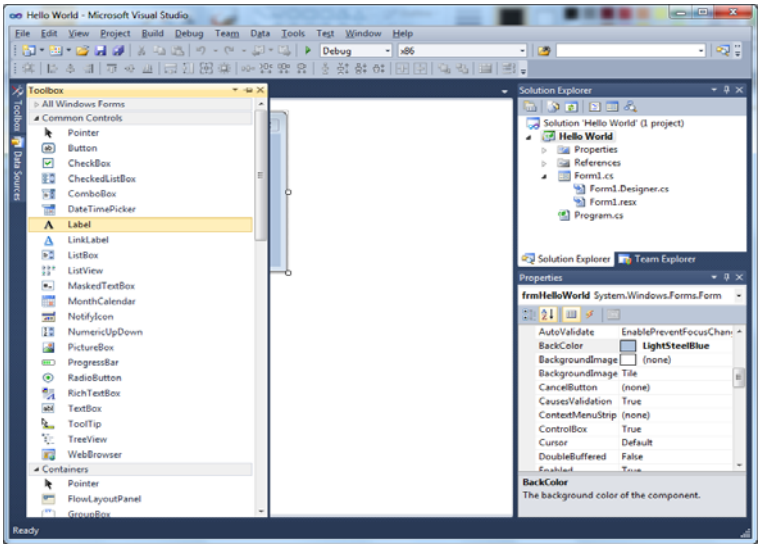

Figure 6C.6 – Placing a Label on the Form

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 **lbScoreLabel.**

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

On the sketch, the label's caption is "**Score**" 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.

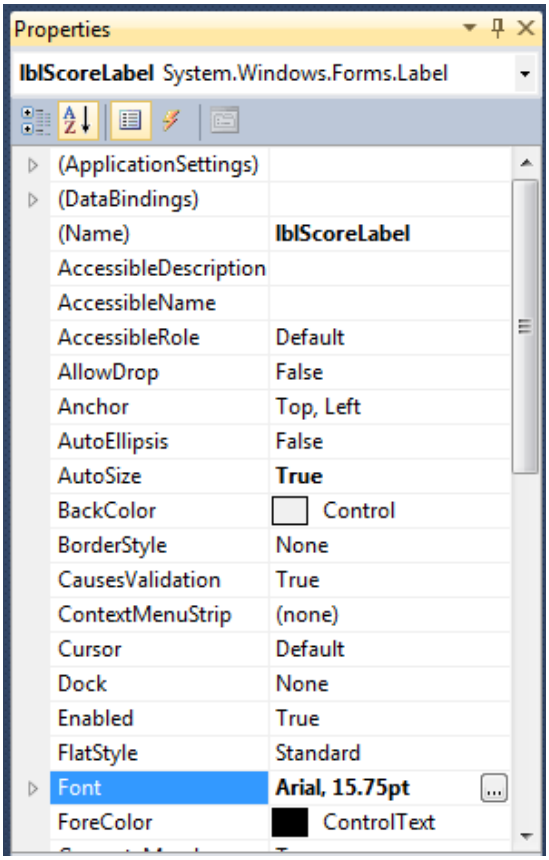

Figure 6C.7 – Changing the Font Property

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.

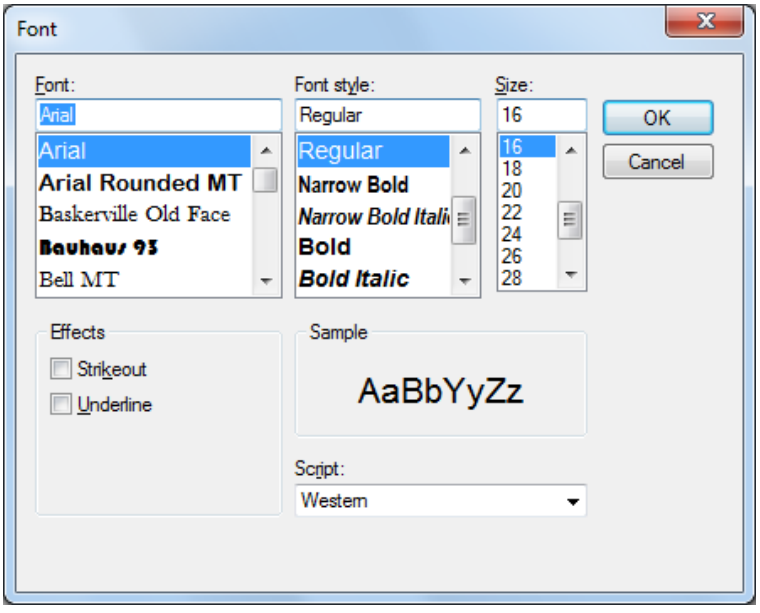

Figure 6C.8 – The Font Window in Visual C#

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.

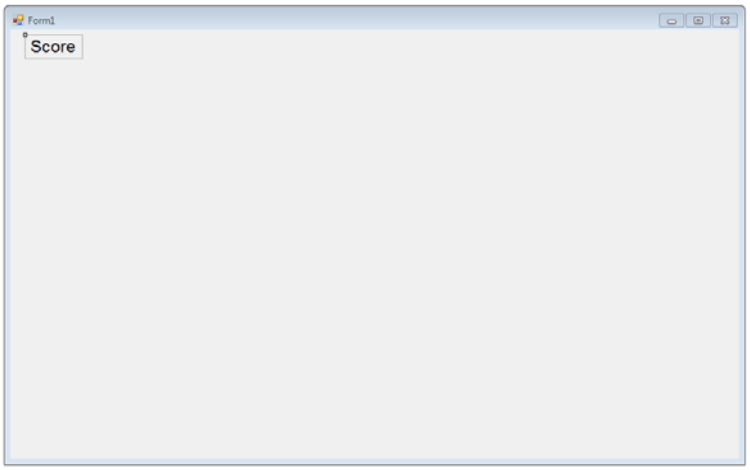

Figure 6C.9 – The Finished Label on the Form

We will call the label for the second player lblLivesLabel and we will make the text state "Lives".

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

The next label is for the name Block Buster which is lblName.

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

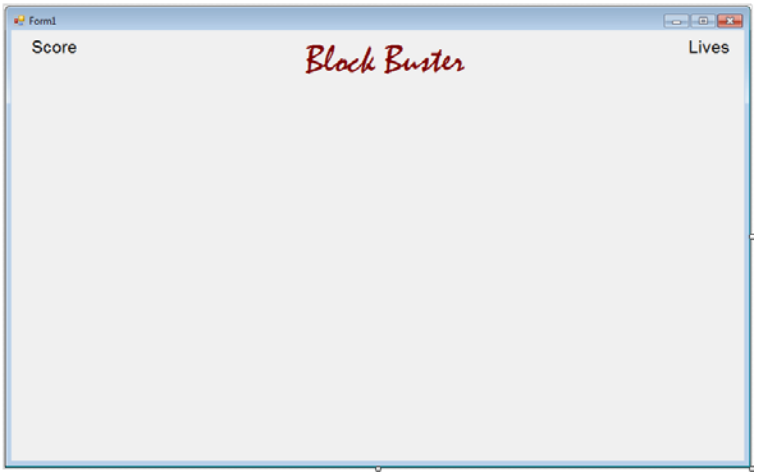

Figure 6C.10 – The Finished Labels on the Form

We will add two labels that will hold the scores and Lives. We will position these labels under their identifiers and they will hold their initial number, 0 for the score and 3 for the number of lives.

Alphabetic	
(Name)	lblScore
AutoSize	False
Border	Fixed Single
BackColor	White
Font	Arial Narrow, 16 pt
Size	54,25
Text	0

Alphabetic	
(Name)	lblLives
AutoSize	False
Border	Fixed Single
BackColor	White
Font	Arial Narrow, 16 pt
Size	54,25
Text	3

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

Figure 6C.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 three pictures on the form, one is for the paddles, the other is the ball and the last is for the block. The paddle is 10 pixels wide by 50 pixels tall, the ball is 10 by 10 pixels and the block is 90 pixel wide and 20 pixels tall. We need to open a graphics program and make both of these icons and save them in our Block buster 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.

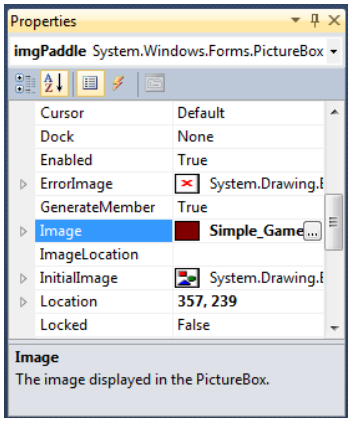

Figure 6C.12 – Adding an Image

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	Verticle_paddle.bmp
Location	430,465
Size	10,50

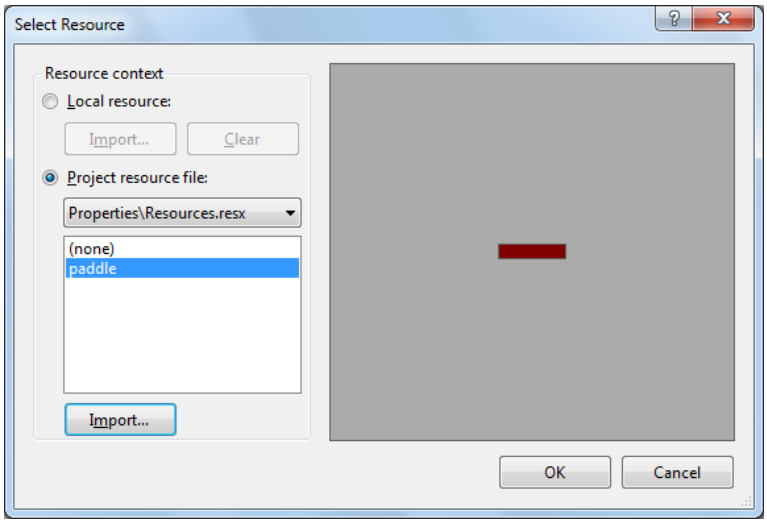

Figure 6C.13 – Import the Paddle Image

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)	imgSphere
Image	Ball.jpg
Location	430,430
Size	10,10

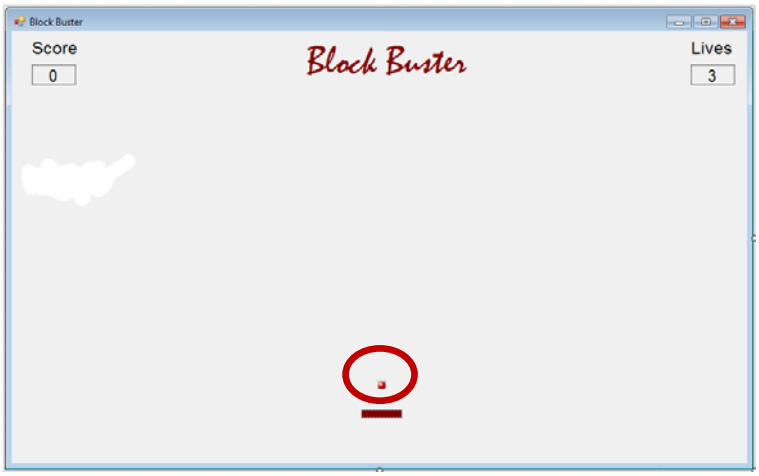

Figure 6C.14 – Import the Ball Image

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)	imgBlockYel1
Image	Block_yel.jpg
Location	20, 173
Size	90,20

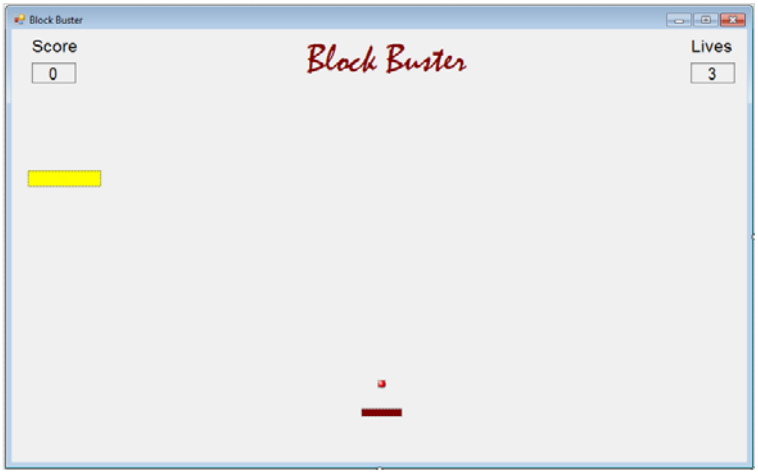

Figure 6C.15 – Import the Block Image

Adding Comments in Visual C# to Communicate to Others

The comments we placed in the first four 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 3
//This program will allow the player to return a ball that will destroy blocks
//to earn points. If the ball passes the paddle the player loses a life up to
//three times
```

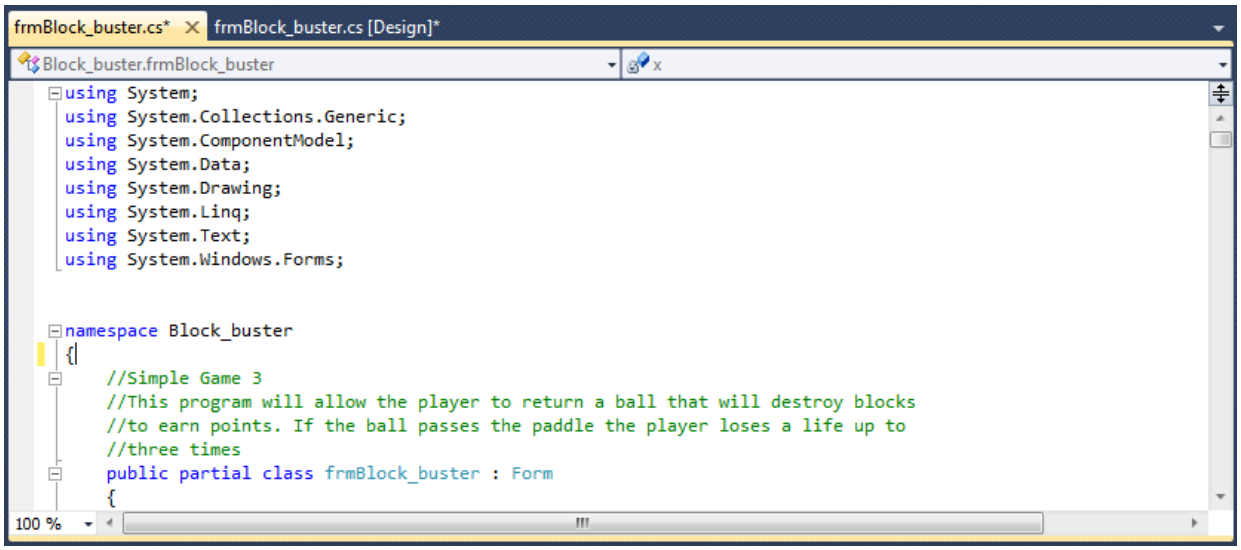

Figure 6C.16 – Adding an Introduction

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, life, blkyel1x and blkyel1y. These variables will hold whole numbers for movement on the form in terms of pixels, so we will declare all ten as integers.

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

```
public partial class frmBlock_buster : 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

    int blkyel1x;   //x coord of yellow block 1
    int blkyel1y;   //y coord of yellow block 1
```

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 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 hits the paddle or a block, so we need a variable for it. Life holds the number 3 and when it reaches zero, we will end the game. The blkyelx1 and blkyel1y hold the origin point for the block. We will add lots of blocks in our game eventually, so we will have more variables. The next block will be blkyelx2 and blkyel12 and so on.

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

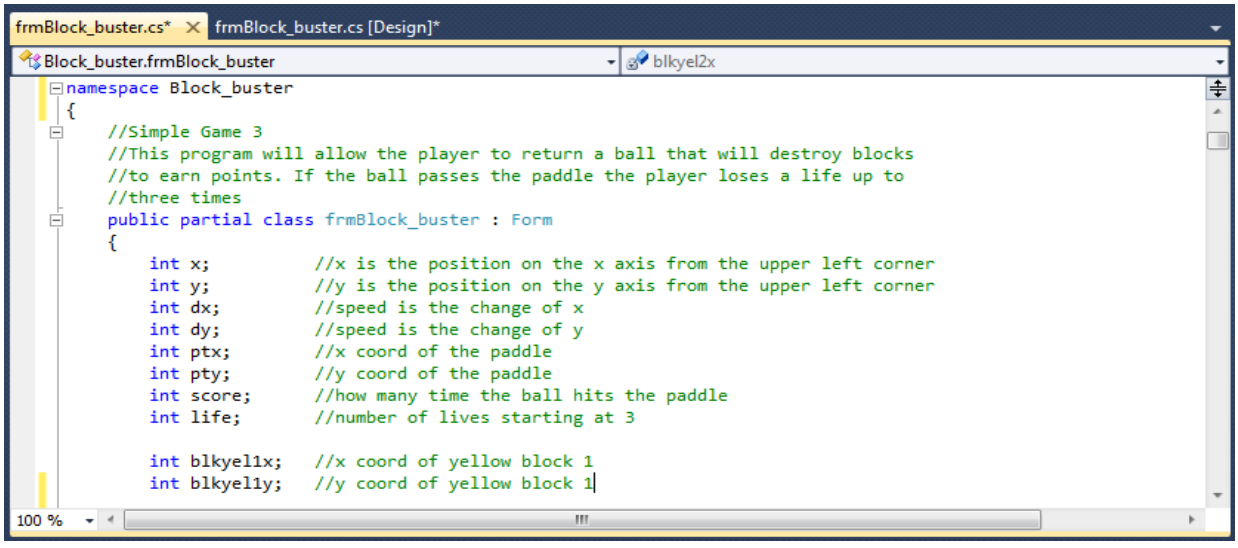

Figure 6C.17 – Declaring Variables with Int Statements

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 block_buster_Load (object sender, EventArgs e) subroutine of the program.

```
private void block_buster_Load(object sender, EventArgs e)
    {
      //Loads the ball on the screen at the upper left corner of the window
      x = 450;
      y = 430;
      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 920 pixels by 569 pixels for our project.

We start the ball at x equals 450 and y equals 430. We keep the change of x (dx) and the change of y (dy) the same throughout the entire program at -1 or -1. The score is set at 0 and the life variable is set at 3.

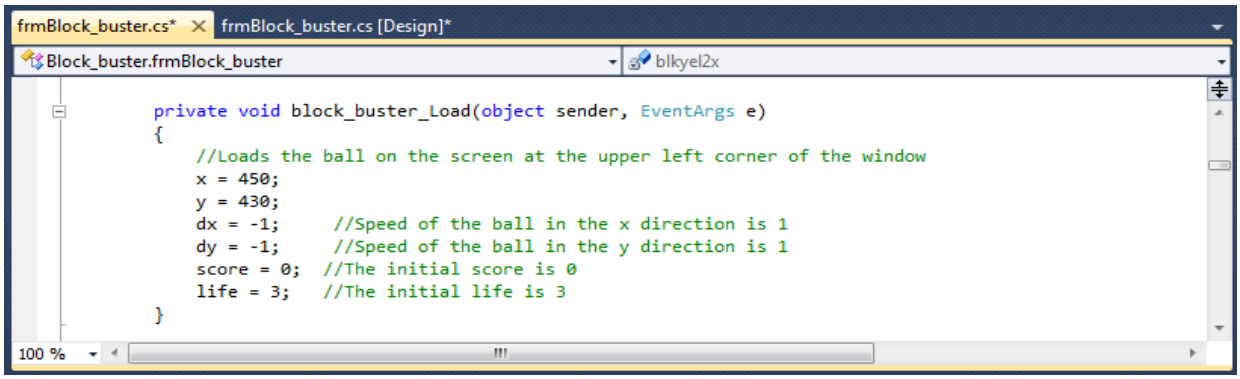

Figure 6C.18 – Setting the Variables in the C# Code

Adding a Timer to the Program

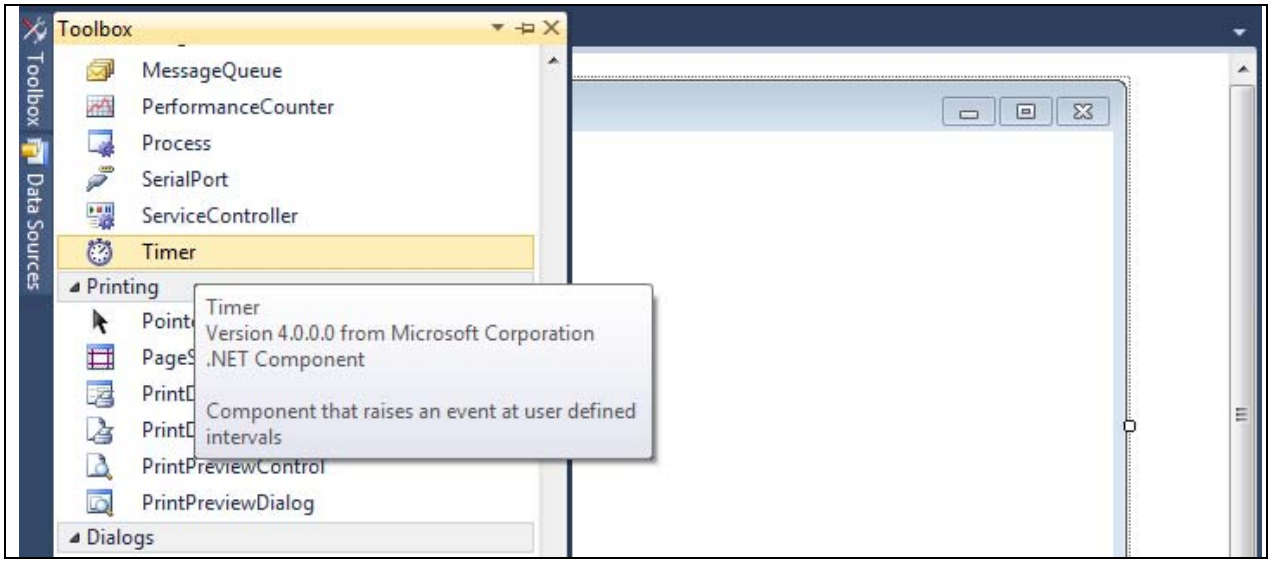

Figure 6C.19 – Adding a Timer

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.

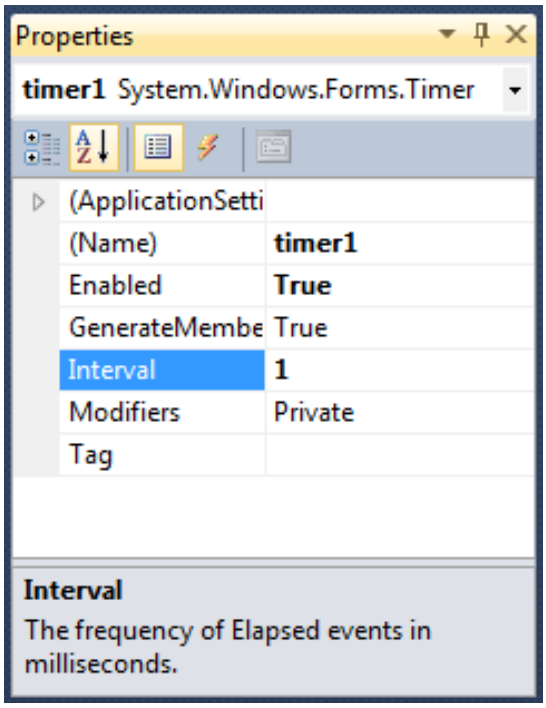

Figure 6C.20 – Setting the Timer's Properties

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)
    {
        imgSphere.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);
        }
    }
}
```

We will now check whether the ball is intersecting with a visible block. We check to see if the block is visible, if the ball is within the left edge of the block, if the ball is within the right edge of the 90 pixel wide block, if the ball is less than the bottom side of the block and if the ball is greater than the top side of the block. If all of the conditions are met, we turn the block to invisible, change the direction if the dy variable, and add one to the score.

```
Point blkpt1 = this.imgBlockYel1.Location;
blkyel1x = (blkpt1.X);
blkyel1y = (blkpt1.Y);
if (imgBlockYel1.Visible == true)
{
    if ((x + 5) >= blkyel1x)
    {
        if ((x + 5) <= (blkyel1x + 90))
        {
            if (y < (blkyel1y + 18))
            {
                if (y > blkyel1y)
                {
                    imgBlockYel1.Visible = false;
                    dy = -dy;
                    score = score + 1;
                    lblScore.Text = Convert.ToString(score);
                }
            }
        }
    }
}
```

If the ball goes by the paddle is this part of the application, we set the ball to the original settings and subtract 1 from the life. If we are at zero lives, then the game is over and x and y will be set to -10 and -10. The expression this.Invalidate(); will refresh the window each click of the timer.

```
if ((y + 9) > pty)
{
    dx = -1;
    dy = -1;
```

```
            x = 450;
            y = 430;
            life = life - 1;
            lblLives.Text = Convert.ToString(life);
        }
        if (life == 0)
        {
            lblGameOver.Text = "Game Over";
            x = -10;
            y = -10;
        }

        this.Invalidate();
    }
```

Double click on the block to see this in our code.

```
    private void imgBlockYel1_Click(object sender, EventArgs e)
    {
    }
```

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 frmBlock_buster() portion of the code and right below InitializeComponent();.

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

Then we add the frmBlock_buster_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. If the left arrow is pressed, the paddle moves 5 pixels to the left. If the right arrow is pressed, the paddle moves 5 pixels right.

The last if statement keeps the paddles 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 frmBlock_buster()
    {
        InitializeComponent();
        KeyDown += new KeyEventHandler(frmBlock_buster_KeyDown);
    }

private void frmBlock_buster_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);
}
```

After running the program, we will insert all eight blocks. Then we will add blocks in depth to build 5 blocks in the column. This will take an hour and we will have lots of practice with copying images, code and renaming variables, but when we are done, we will have a fun application.

Running the Program

After noting that the program is saved, press the F5 to run the Block buster application. The Block buster 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 Block buster program, click on the red X in the upper right corner to close the application.

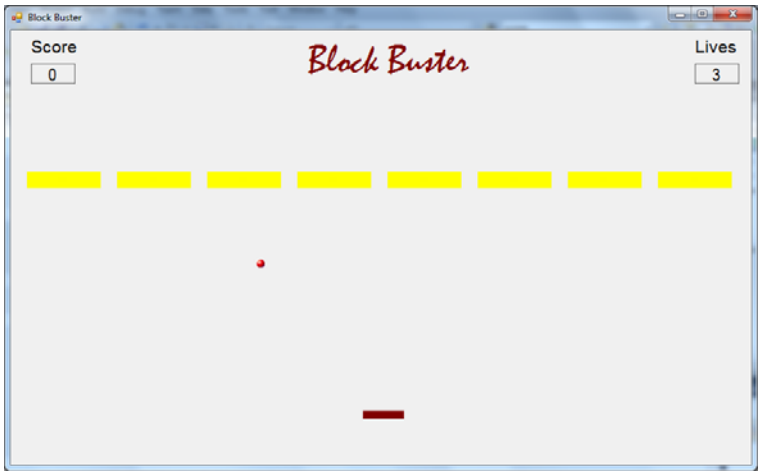

Figure 6C.21 – Launching the Program

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.

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 90-11 * - 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.