

Bonus Projects



Preview

By now, you should feel pretty comfortable with the steps involved in building a Visual C# project. In this bonus chapter, we give you more projects you can build and try. We'll present the steps involved in building each project - **Project Design, Place Controls on Form, Set Control Properties, Write Event Methods, Run the Project, and Other Things to Try**. But, we won't give you detailed discussion of what's going on in the code (we will point out new ideas). You should be able to figure that out by now (with the help of the code comments). Actually, a very valuable programming skill to have is the ability to read and understand someone else's code.

The twelve new projects included are: Computer Stopwatch, Times Tables, Dice Rolling, State Capitals, Memory Game, Unit Conversions, Decode, Frown and three college prep programs: Loan Calculator, Checkbook Balancer, and Portfolio Manager. And, as a bonus, we'll throw in a Visual C# version of the first video game ever – Pong!

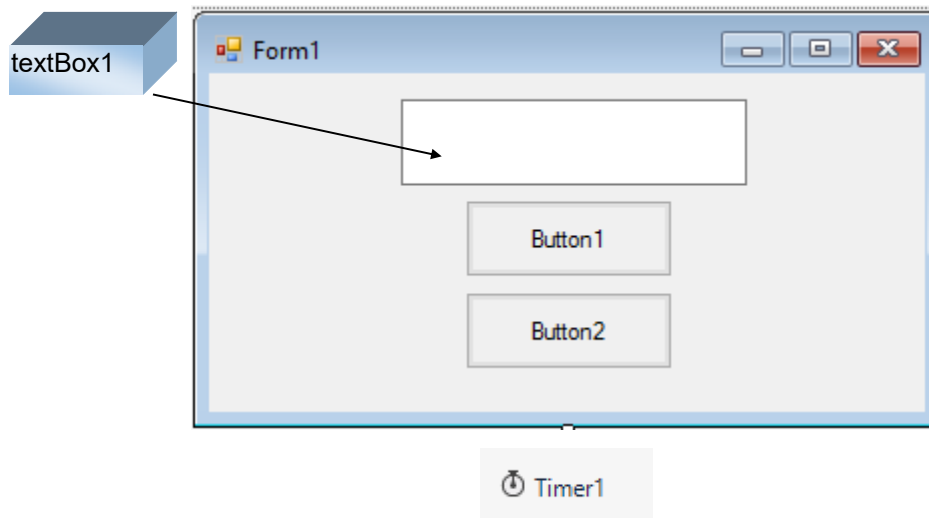
Project 1 - Computer Stopwatch

Project Design

In this project, we will build a computer stopwatch that measures elapsed time in seconds. One button will start and stop the timing and one will reset the display (a label). Elapsed time is measured using the C# **Now** function that provides the current time and date in a **Date** type function. The project you are about to build is saved as **Stopwatch** in the project folder (\BeginVCS\BVCS Projects).

Place Controls on Form

Start a new project in Visual C#. Place a text box control on the form. Then place two buttons on the form. Add a timer control. When done, your form should look something like this:



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Stopwatch
FormBorderStyle	Fixed Single
StartPosition	CenterScreen

textBox1 Text Box:

Property Name	Property Value
Name	txtTime
Text	00:00:00
BackColor	White
Font	Arial
Font Size	24
Font Style	Bold
ReadOnly	True
TextAlign	Center
TabStop	False

button1 Button:

Property Name	Property Value
Name	btnStartStop
Text	Start
Font	Arial
Font Size	12

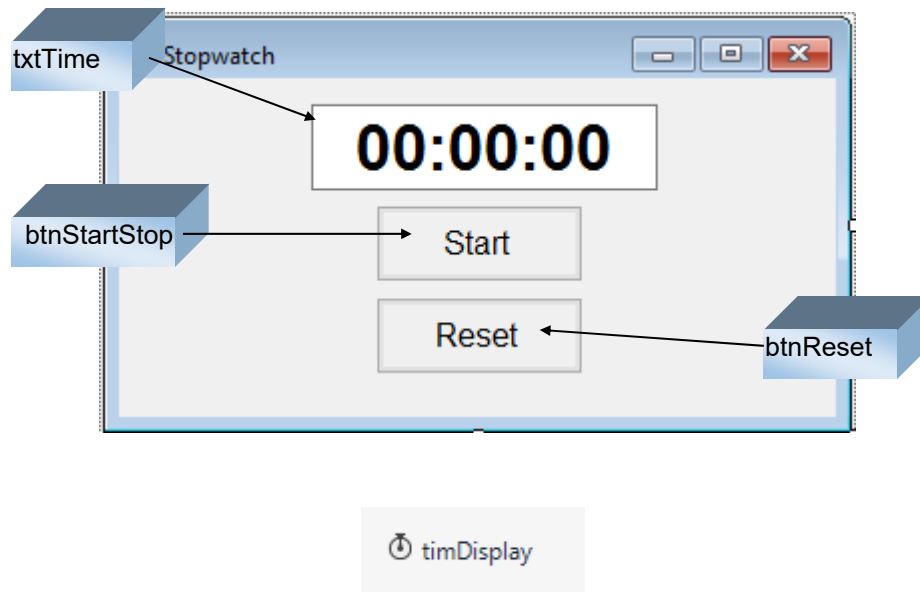
button2 Command Button:

Property Name	Property Value
Name	btnReset
Text	Reset
Enabled	False
Font	Arial
Font Size	12

timer1 Timer:

Property Name	Property Value
Name	timDisplay
Interval	1000

When done setting properties, my form looks like this (I resized the text box a bit):



Write Event methods

To start the stopwatch, click **Start**. To stop, click **Stop**. Click **Reset** to reset the display to zero. Each of these buttons has a **Click** event. The timer control **Tick** event controls the display of the time.

Add this code to the **general declarations** area:

```
DateTime startTime; // time when stopwatch started
```

The **btnStartStop_Click** event method:

```
private void btnStartStop_Click(object sender, EventArgs e)
{
    // Starting timer?
    if (btnStartStop.Text == "Start")
    {
        // Reset Text on Start/Stop button
        btnStartStop.Text = "Stop";
        // Start timer and get starting time
        timDisplay.Enabled = true;
        startTime = DateTime.Now;
    }
    else
    {
        // Stop timer
        timDisplay.Enabled = false;
        // Disable Start/Stop button, enable Reset button
        btnStartStop.Enabled = false;
        btnReset.Enabled = true;
    }
}
```

The **btnReset_Click** event method:

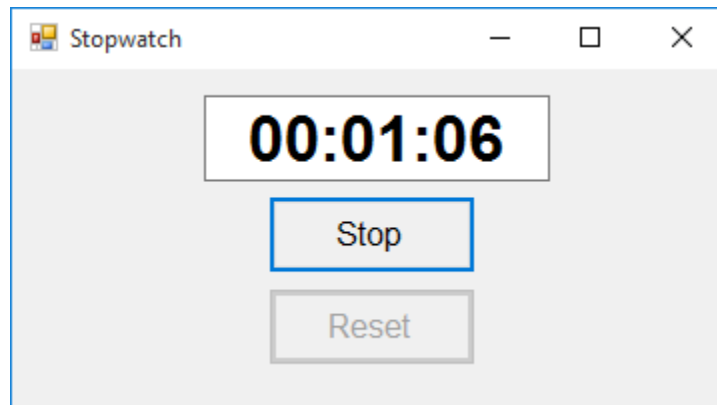
```
private void btnReset_Click(object sender, EventArgs e)
{
    // Reset display to zero
    txtTime.Text = "00:00:00";
    // Reset button Text and enable Start, disable Reset
    btnStartStop.Text = "Start";
    btnStartStop.Enabled = true;
    btnReset.Enabled = false;
}
```

The **timDisplay_Tick** event method:

```
private void timDisplay_Tick(object sender, EventArgs e)
{
    TimeSpan elapsedTime;
    // Determine elapsed time since Start was clicked
    elapsedTime = DateTime.Now - startTime;
    // Display time in label box
    txtTime.Text = Convert.ToString(new
TimeSpan(elapsedTime.Hours, elapsedTime.Minutes,
elapsedTime.Seconds));
}
```

Run the Project

Save your work. Run the project. Click **Start** to start the timer. Make sure the display updates every second. Here's a run I made:



Study the Tick event if you're unsure of how this is done – especially look at how to subtract two times, **DateTime** types, using the C# **TimeSpan** type to get the elapsed time. Click **Stop** to stop the timer. Make sure the **Reset** button works properly.

Other Things to Try

Many stopwatches allow you to continue timing after you've stopped one or more times. That is, you can measure total elapsed time in different segments. Modify this project to allow such measurement. You'll need a separate Stop button and a variable to keep track of total elapsed time. You'll also need to determine which buttons you want to have enabled at different times in the project. Add a "lap timing" feature by displaying the time measured in each segment (a segment being defined as the time between each Start and Stop click).

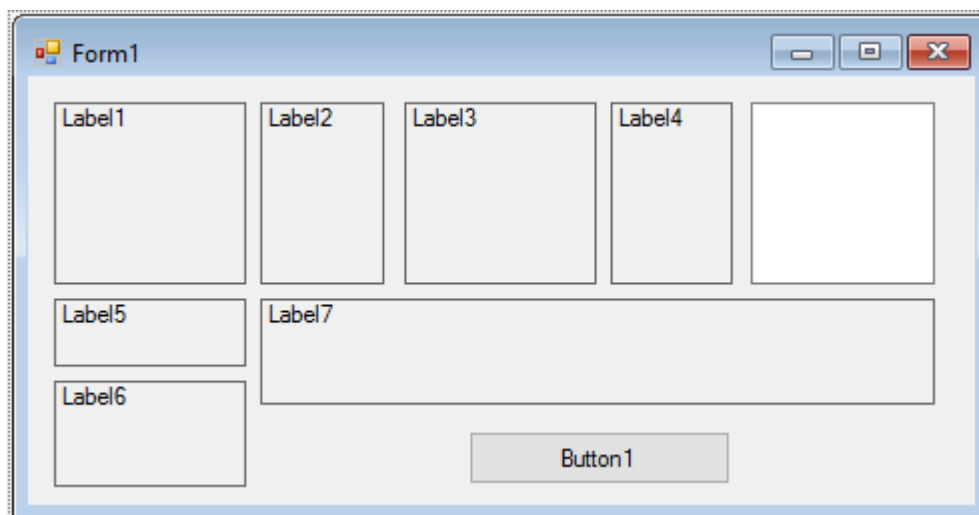
Project 2 – Times Tables

Project Design

In this project, you can give a child practice with the times tables using the numbers from 0 to 9. The computer generates a random problem. The child answers and the computer evaluates the performance. The project you are about to build is saved as **Times** in the project folder (**\\BeginVCS\\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Place seven labels (with **AutoSize** set to **False**, to allow resizing), a text box and a button on the form. When done, your form should look something like this (I've temporarily set the border style of each label control to **FixedSingle** to show placement; you might also like to do this, but remember to change border style back to **None**):



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Times Tables
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

label1 Label:

Property Name	Property Value
Name	lblNum1
Text	[Blank]
TextAlign	MiddleCenter
Font	Arial
Font Size	48
AutoSize	False

label2 Label:

Property Name	Property Value
Text	x
TextAlign	MiddleCenter
Font	Arial
Font Size	48
AutoSize	False

label3 Label:

Property Name	Property Value
Name	lblNum2
Text	[Blank]
TextAlign	MiddleCenter
Font	Arial
Font Size	48
AutoSize	False

label4 Label:

Property Name	Property Value
Text	=
TextAlign	MiddleCenter
Font	Arial
Font Size	48
AutoSize	False

label5 Label:

Property Name	Property Value
Text	Score:
TextAlign	MiddleCenter
Font Size	18
AutoSize	False

label6 Label:

Property Name	Property Value
Name	lblScore
Text	0%
TextAlign	MiddleCenter
BackColor	Light Yellow
BorderStyle	Fixed3D
Font Size	20
AutoSize	False

label7 Label:

Property Name	Property Value
Name	lblMessage
Text	[Blank]
TextAlign	MiddleCenter
BackColor	Light Yellow
BorderStyle	Fixed3D
Font Size	24
AutoSize	False

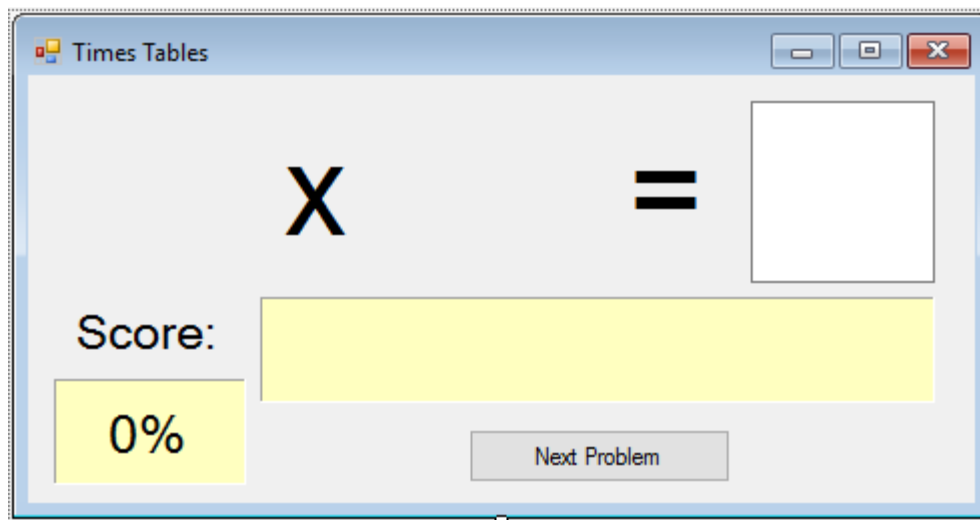
textBox1 Text Box:

Property Name	Property Value
Name	txtAnswer
Text	[Blank]
TextAlign	Center
Font	Arial
Font Size	48
MaxLength	2

button1 Button:

Property Name	Property Value
Name	btnNext
Text	Next Problem

When done setting properties, my form looks like this:



Write Event Methods

When the user clicks **Next Problem**, the computer generates and displays a multiplication problem. The user types an answer and presses **<Enter>**. If correct, you are told so. If incorrect, the correct answer is given. In either case, the score is updated. Continue answering as long as you would like.

Add this code to the **general declarations** area:

```
int product;  
int numProb;  
int numRight;  
Random myRandom = new Random();
```

The **Form1_Load** event method:

```
private void Form1_Load(object sender, EventArgs e)  
{  
    // Initialize variables  
    numProb = 0;  
    numRight = 0;  
    // display the first problem  
    btnNext.PerformClick();  
}
```

The `btnNext_Click` event method:

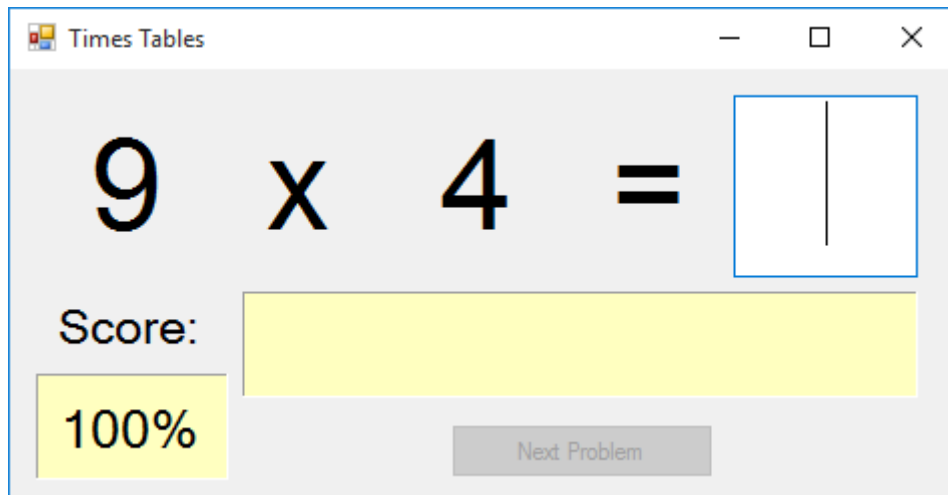
```
private void btnNext_Click(object sender, EventArgs e)
{
    // Generate next multiplication problem
    int number1, number2;
    txtAnswer.Text = "";
    lblMessage.Text = "";
    numProb++;
    // Generate random numbers for factors
    number1 = myRandom.Next(10);
    number2 = myRandom.Next(10);
    lblNum1.Text = Convert.ToString(number1);
    lblNum2.Text = Convert.ToString(number2);
    // Find product
    product = number1 * number2;
    btnNext.Enabled = false;
    txtAnswer.Focus();
}
```

The `txtAnswer_KeyPress` event method:

```
private void txtAnswer_KeyPress(object sender,
KeyPressEventArgs e)
{
    int ans;
    // Check for number only input and for return key
    if ((e.KeyChar >= '0' && e.KeyChar <= '9') || (int)
e.KeyChar == 8)
    {
        e.Handled = false;
    }
    else if ((int) e.KeyChar == 13)
    {
        // Check answer and update score
        ans = Convert.ToInt32(txtAnswer.Text);
        if (ans == product)
        {
            numRight++;
            lblMessage.Text = "That's correct!";
        }
        else
        {
            lblMessage.Text = "Answer is " +
Convert.ToString(product);
        }
        lblScore.Text = String.Format("{0:f0}", 100 *
((double) numRight / numProb)) + "%";
        btnNext.Enabled = true;
        btnNext.Focus();
    }
    else
    {
        e.Handled = true;
    }
}
```

Run the Project

Save your work. Run the project. A multiplication problem will be displayed. Type an answer and press <Enter>. If correct, that's great. If not, you will be shown the correct answer. Click **Next Problem** for another problem. Try for a high score. Here's a run I made:



Other Things to Try

Some suggested changes to make this a more useful program are: (1) make the range of factors an option (small numbers for little kids, large numbers for older kids), (2) allow practice with a specific factor only, (3) give the user more chances at the correct answer with a decreasing score for each try, (4) set up a timer so the faster the user answers, the higher the score and (5) expand the program to include other operations such as addition, subtraction and division.

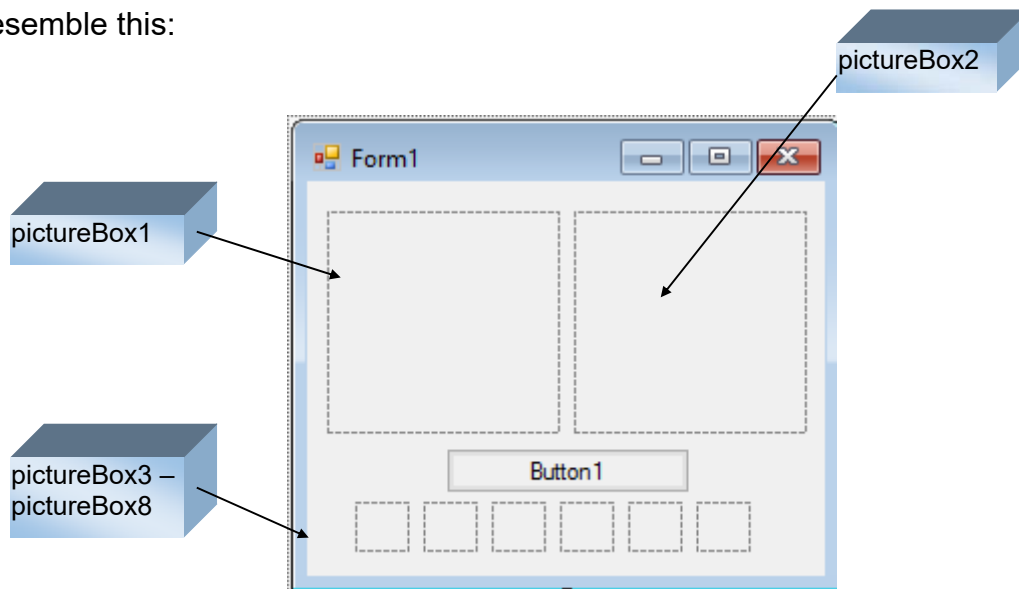
Project 3 – Dice Rolling

Project Design

It happens all the time. You get your favorite game out and the dice are missing! This program comes to the rescue – it uses the C# random number generator to roll two dice for you. Simply click a button to see the two dice displayed. A group of picture box controls will hold the six possible die values. This project is saved as **DiceRoll** in the project folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Place two large picture box controls (to display the dice) and six small picture box controls (to hold the six possible die pictures) on the form. Place one button on the form. When done, your form should resemble this:



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Dice Rolling
BackColor	Red
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

pictureBox1 Picture Box:

Property Name	Property Value
Name	picDice1
SizeMode	StretchImage

pictureBox2 Picture Box:

Property Name	Property Value
Name	picDice2
SizeMode	StretchImage

pictureBox3 Picture Box:

Property Name	Property Value
Name	picDots1
Image	Dice1.gif (in \VCSKids\VCSK Projects\DiceRoll folder)
SizeMode	StretchImage
Visible	False

pictureBox4 Picture Box:

Property Name	Property Value
Name	picDots2
Image	Dice2.gif (in \VCSKids\VCSK Projects\DiceRoll folder)
SizeMode	StretchImage
Visible	False

pictureBox5 Picture Box:

Property Name	Property Value
Name	picDots3
Image	Dice3.gif (in \VCSKids\VCSK Projects\DiceRoll folder)
SizeMode	StretchImage
Visible	False

pictureBox6 Picture Box:

Property Name	Property Value
Name	picDots4
Image	Dice4.gif (in \VCSKids\VCSK Projects\DiceRoll folder)
SizeMode	StretchImage
Visible	False

pictureBox7 Picture Box:

Property Name	Property Value
Name	picDots5
Image	Dice5.gif (in \VCSKids\VCSK Projects\DiceRoll folder)
SizeMode	StretchImage
Visible	False

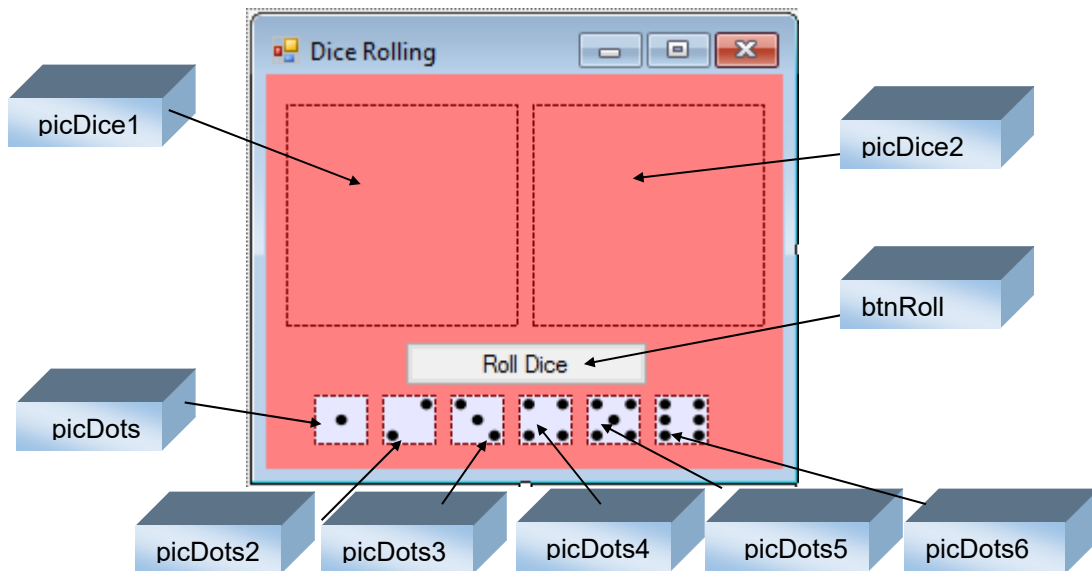
pictureBox8 Picture Box:

Property Name	Property Value
Name	picDots6
Image	Dice6.gif (in \VCSKids\VCSK Projects\DiceRoll folder)
SizeMode	StretchImage
Visible	False

button1 Button:

Property Name	Property Value
Name	btnRoll
Text	Roll Dice

When, done my form looks like this:



Notice we use two sets of picture boxes. The first, `picDice1` and `picDice2`, is used to display the two dice. The second, `picDots1` – `picDots6`, is used to store the six possible die pictures. This second group has a `Visible` property of `False`. Hence, you only see them displayed at design time.

Write Event methods

To roll the dice, simply click **Roll Dice**.

Declare an array of images (named **dots**) in the **general declarations** area. This array will be used to choose which of the six possible images to display. You also need a random number object:

```
Image[] dots = new Image[6];  
Random myRandom = new Random();
```

Add this code to the **Form1_Load** event. Here, we establish the image array and 'click' the **btnRoll** button to 'roll' the dice before the display is activated:

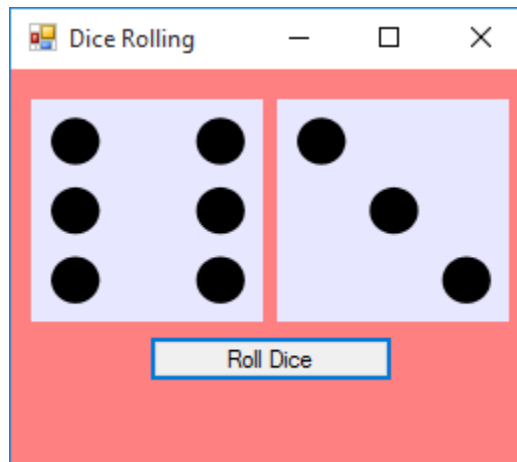
```
private void Form1_Load(object sender, EventArgs e)  
{  
    // initialize display  
    dots[0] = picDots1.Image;  
    dots[1] = picDots2.Image;  
    dots[2] = picDots3.Image;  
    dots[3] = picDots4.Image;  
    dots[4] = picDots5.Image;  
    dots[5] = picDots6.Image;  
    btnRoll.PerformClick();  
}
```

The **btnRoll_Click** event method:

```
private void btnRoll_Click(object sender, EventArgs e)  
{  
    // Roll Dice 1 and set display  
    picDice1.Image = dots[myRandom.Next(6)];  
    // Roll Dice 2 and set display  
    picDice2.Image = dots[myRandom.Next(6)];  
}
```

Run the Project

Save your work. Run the project. Click **Roll Dice** to see the dice change with each click. Look at the code to see how the random number (1 through 6) is generated and how the image array (**Dots**) sets the display. Here's one of my rolls:



Other Things to Try

The game of Yahtzee requires 5 dice. Modify the project to roll and display five dice. Or, let the user decide how many dice to display (you could more 'display' picture boxes and use the **Visible** property to specify whether a particular die is displayed). Add a label control that displays the sum of the displayed dice.

A fun change would be to have the die displays updated by a **Timer** control to give the appearance of rolling dice. You would need a Timer control for each die (every 100 milliseconds or so, randomly display from 1 to 6 dots). And, then you would need a Timer control to stop the 'rolling' (use an **Interval** of about 2000 milliseconds). The **btnRoll** button would control enabling on the Timer controls. All Timer controls are turned off (Enabled is set to false) by the Timer event that stops the rolling.

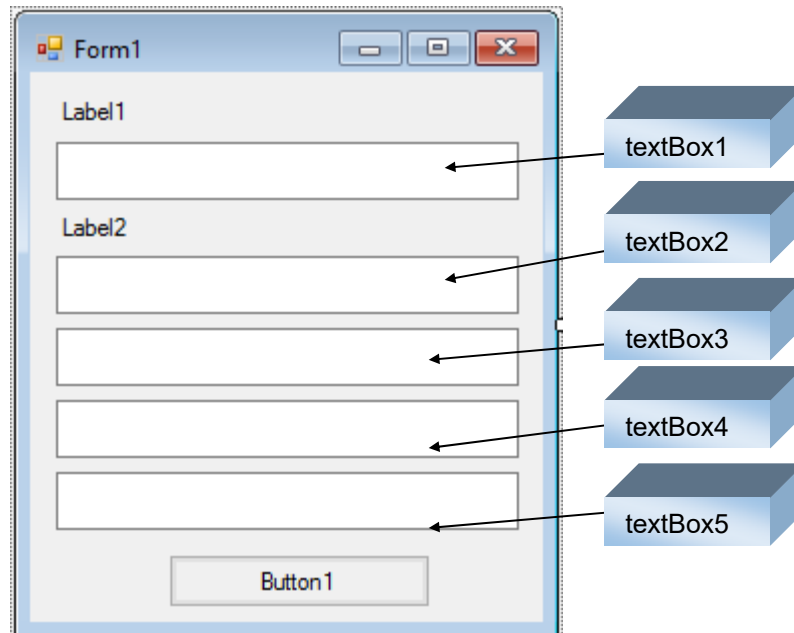
Project 4 – State Capitals

Project Design

In this project, we build a fun game for home and school. You will be given the name of a state in the United States and four possible choices for its capital city. You click on the guess of your choice to see if you are right. (We apologize to our foreign readers – perhaps you can modify this project to build a similar multiple choice type game). Click on **Next State** for another question. This project is saved as **StateCapitals** in the project folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Place two label controls, five text boxes and two buttons on the form. When done, your form should resemble this:



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	State Capitals
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

label1 Label:

Property Name	Property Value
Name	lblHeadState
Text	State:
Font Size	14
Font Style	Italic

textBox1 Text Box:

Property Name	Property Value
Name	txtState
BackColor	White
Font Size	14
ReadOnly	True
TextAlign	Center
TabStop	False

label2 Label:

Property Name	Property Value
Name	lblHeadCapital
Text	Capital:
Font Size	14
Font Style	Italic

textBox2 Text Box:

Property Name	Property Value
Name	txtCapital0
BackColor	White
Font Size	14
ReadOnly	True
TextAlign	Center
TabStop	False

textBox3 Text Box:

Property Name	Property Value
Name	txtCapital1
BackColor	White
Font Size	14
ReadOnly	True
TextAlign	Center
TabStop	False

textBox4 Text Box:

Property Name	Property Value
Name	txtCapital2
BackColor	White
Font Size	14
ReadOnly	True
TextAlign	Center
TabStop	False

textBox5 Text Box:

Property Name	Property Value
Name	txtCapita3
BackColor	White
Font Size	14
ReadOnly	True
TextAlign	Center
TabStop	False

button1 Button:

Property Name	Property Value
Name	btnNext
Text	Next State

button2 Button:

Property Name	Property Value
Name	btnCheck

Resize the form so **btnCheck** does not appear (we'll use this as a general method button). When done, the form looks like this :

The screenshot shows a Windows-style window titled "State Capitals". Inside the window, there are two main sections. The first section is labeled "State:" in a bold, italicized font, followed by a single text input field. The second section is labeled "Capital:" in a bold, italicized font, followed by four stacked text input fields. At the bottom of the window is a button labeled "Next Capital".

External labels with arrows point to the following components:

- lblHeadState** points to the "State:" label.
- lblHeadCapital** points to the "Capital:" label.
- txtState** points to the text input field under "State:".
- txtCapital0** points to the first text input field under "Capital:".
- txtCapital1** points to the second text input field under "Capital:".
- txtCapital2** points to the third text input field under "Capital:".
- txtCapital3** points to the fourth text input field under "Capital:".
- btnNext** points to the "Next Capital" button.

Write Event methods

To display a state and possible capitals, click **Next State**. Click on your choice for answer.

Put this code in the general declarations area:

```
int answer;
string[] state = new string[50];
string[] capital = new string[50];
TextBox[] listedCapital = new TextBox[4];
int capitalClicked;
Random myRandom = new Random();
```

Add this code to the **Form1_Load** event (yes, it's a lot of typing or just copy and paste from these notes):

```
private void Form1_Load(object sender, EventArgs e)
{
    // load state and capital arrays
    state[0] = "Alabama"; capital[0] = "Montgomery";
    state[1] = "Alaska"; capital[1] = "Juneau";
    state[2] = "Arizona"; capital[2] = "Phoenix";
    state[3] = "Arkansas"; capital[3] = "Little Rock";
    state[4] = "California"; capital[4] = "Sacramento";
    state[5] = "Colorado"; capital[5] = "Denver";
    state[6] = "Connecticut"; capital[6] = "Hartford";
    state[7] = "Delaware"; capital[7] = "Dover";
    state[8] = "Florida"; capital[8] = "Tallahassee";
    state[9] = "Georgia"; capital[9] = "Atlanta";
    state[10] = "Hawaii"; capital[10] = "Honolulu";
    state[11] = "Idaho"; capital[11] = "Boise";
    state[12] = "Illinois"; capital[12] = "Springfield";
    state[13] = "Indiana"; capital[13] = "Indianapolis";
    state[14] = "Iowa"; capital[14] = "Des Moines";
    state[15] = "Kansas"; capital[15] = "Topeka";
    state[16] = "Kentucky"; capital[16] = "Frankfort";
    state[17] = "Louisiana"; capital[17] = "Baton Rouge";
    state[18] = "Maine"; capital[18] = "Augusta";
    state[19] = "Maryland"; capital[19] = "Annapolis";
```

```
state[20] = "Massachusetts"; capital[20] = "Boston";
state[21] = "Michigan"; capital[21] = "Lansing";
state[22] = "Minnesota"; capital[22] = "Saint Paul";
state[23] = "Mississippi"; capital[23] = "Jackson";
state[24] = "Missouri"; capital[24] = "Jefferson City";
state[25] = "Montana"; capital[25] = "Helena";
state[26] = "Nebraska"; capital[26] = "Lincoln";
state[27] = "Nevada"; capital[27] = "Carson City";
state[28] = "New Hampshire"; capital[28] = "Concord";
state[29] = "New Jersey"; capital[29] = "Trenton";
state[30] = "New Mexico"; capital[30] = "Santa Fe";
state[31] = "New York"; capital[31] = "Albany";
state[32] = "North Carolina"; capital[32] = "Raleigh";
state[33] = "North Dakota"; capital[33] = "Bismarck";
state[34] = "Ohio"; capital[34] = "Columbus";
state[35] = "Oklahoma"; capital[35] = "Oklahoma City";
state[36] = "Oregon"; capital[36] = "Salem";
state[37] = "Pennsylvania"; capital[37] = "Harrisburg";
state[38] = "Rhode Island"; capital[38] = "Providence";
state[39] = "South Carolina"; capital[39] = "Columbia";
state[40] = "South Dakota"; capital[40] = "Pierre";
state[41] = "Tennessee"; capital[41] = "Nashville";
state[42] = "Texas"; capital[42] = "Austin";
state[43] = "Utah"; capital[43] = "Salt Lake City";
state[44] = "Vermont"; capital[44] = "Montpelier";
state[45] = "Virginia"; capital[45] = "Richmond";
state[46] = "Washington"; capital[46] = "Olympia";
state[47] = "West Virginia"; capital[47] = "Charleston";
state[48] = "Wisconsin"; capital[48] = "Madison";
state[49] = "Wyoming"; capital[49] = "Cheyenne";
// Set listed capital labels
listedCapital[0] = txtCapital0;
listedCapital[1] = txtCapital1;
listedCapital[2] = txtCapital2;
listedCapital[3] = txtCapital3;
// set first question
btnNext.PerformClick();
}
```

The **btnNext_Click** event method generates the next multiple choice question:

```
private void btnNext_Click(object sender, EventArgs e)
{
    int[] vUsed = new int[50];
    int[] index = new int[4];
    int j;
    // Generate the next question at random
    btnNext.Enabled = false;
    answer = myRandom.Next(50);
    // Display selected state
    txtState.Text = state[answer];
    // Vused array is used to see which state capitals have
    // been selected as possible answers
    for (int i = 0; i < 50; i++)
    {
        vUsed[i] = 0;
    }
    // Pick four different state indices (J) at random
    // These are used to set up multiple choice answers
    // Stored in the Index array
    for (int i = 0; i < 4; i++)
    {
        // Find index not used yet and not the answer
        //Find value not used yet and not the answer
        do
        {
            j = myRandom.Next(50);
        }
        while (vUsed[j] != 0 || j == answer);
        vUsed[j] = 1;
        index[i] = j;
    }
    // Now replace one index (at random) with correct answer
    index[myRandom.Next(4)] = answer;
    // Display multiple choice answers in text boxes
    for (int i = 0; i < 4; i++)
    {
        listedCapital[i].Text = capital[index[i]];
    }
}
```

A new concept in this routine is the **do** loop (shaded line) to pick the different possible answers. Let's explain how this particular loop works.

The form used in this code is:

```
do
{
    [C# code]
}
while condition;
```

In this "loop," the C# code between the **do** line and the **while** line is repeated as long as the specified **condition** is true. See if you can see how this loop allows us to pick four distinct capital cities for the multiple choice answers (no repeated values).

The event methods for clicking the capital city text box controls simply identify which text box was clicked and “clicks” the hidden **btnCheck** button:

```
private void txtCapital0_Click(object sender, EventArgs e)
{
    capitalClicked = 0;
    btnCheck.PerformClick();
}

private void txtCapital1_Click(object sender, EventArgs e)
{
    capitalClicked = 1;
    btnCheck.PerformClick();
}

private void txtCapital2_Click(object sender, EventArgs e)
{
    capitalClicked = 2;
    btnCheck.PerformClick();
}

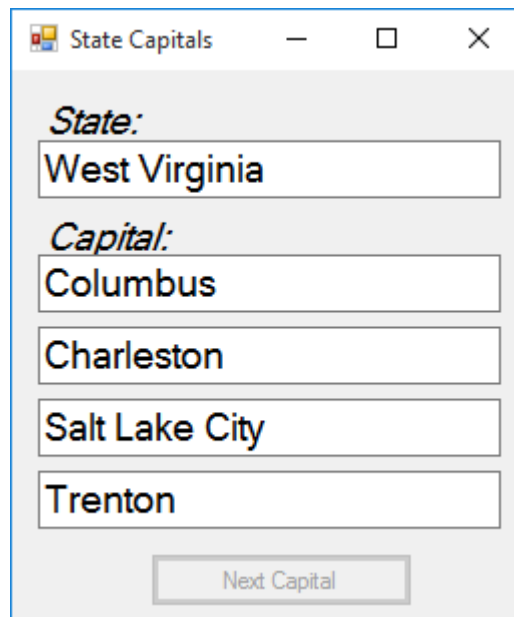
private void txtCapital3_Click(object sender, EventArgs e)
{
    capitalClicked = 3;
    btnCheck.PerformClick();
}
```


We write a **btnCheck_Click** “hidden” general method to check the answer (when it is selected by clicking a text box control):

```
private void btnCheck_Click(object sender, EventArgs e)
{
    // If already answered, ignore click
    if (listedCapital[capitalClicked].Text != "" &&
        btnNext.Enabled == false)
    {
        // Check clicked answer
        if (listedCapital[capitalClicked].Text ==
            capital[answer])
        {
            // Correct answer - clear out other answers and
            enable Next button
            for (int i = 0; i < 4; i++)
            {
                if (i != capitalClicked)
                {
                    listedCapital[i].Text = "";
                }
            }
            btnNext.Enabled = true;
            btnNext.Focus();
        }
        else
        {
            // Incorrect answer - clear out selected answer
            listedCapital[capitalClicked].Text = "";
        }
    }
}
```

Run the Project

Save your work. Run the project. A state name and four possible capital cities will be displayed. (Study the code used to choose and sort the possible answers – this kind of code is very useful.) Choose an answer. If correct, the other answers will be cleared. Click **Next State** to continue. If incorrect, your choice will be cleared. Keeping answering until correct. Here's a run I made where I missed on my first guess:



Other Things to Try

This would be a fun project to modify. How about changing it to display a capital city with four states as the multiple choices? Allow the user to type in the answer (use a text box) instead of picking from a list. Add some kind of scoring system.

This program could also be used to build general multiple choice tests from any two lists. You could do language translations (given a word in English, choose the corresponding word in Spanish), given a book, choose the author, or given an invention, name the inventor. Use your imagination.

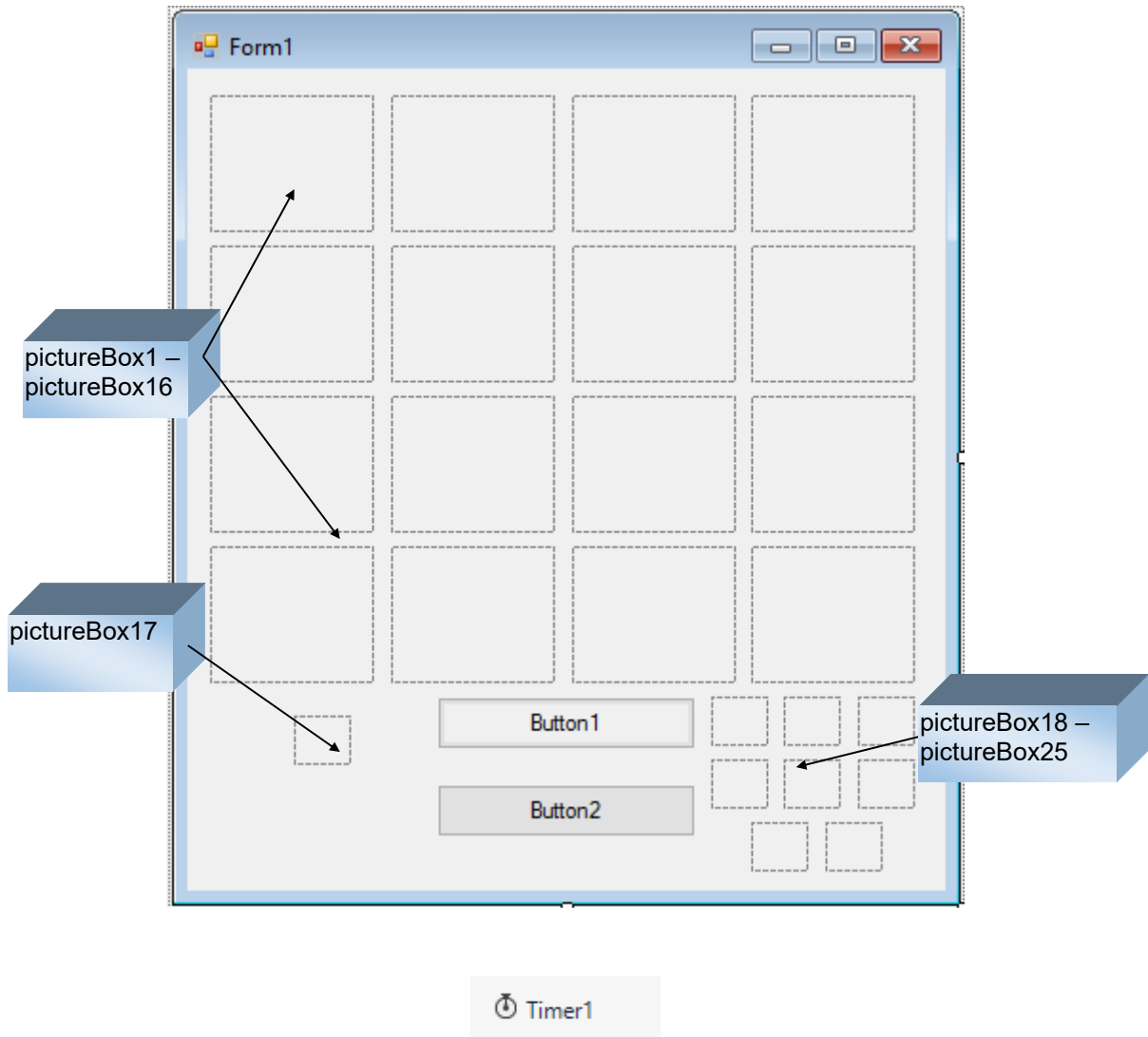
Project 5 – Memory Game

Project Design

In this game for little kids, sixteen squares are used to hide eight different pairs of pictures. The player chooses two squares on the board and the pictures behind them are revealed. If the pictures match, those squares are removed from the board. If there is no match, the pictures are recovered and the player tries again. The play continues until all eight pairs are matched up. The game is saved as **MemoryGame** in the project folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Place sixteen large picture box controls and nine smaller picture box controls on the form. Place two buttons and a timer on the form. When done, your form should resemble this:



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Memory Game
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

pictureBox1 Picture Box:

Property Name	Property Value
Name	picHidden0
SizeMode	StretchImage

pictureBox2 Picture Box:

Property Name	Property Value
Name	picHidden1
SizeMode	StretchImage

pictureBox3 Picture Box:

Property Name	Property Value
Name	picHidden2
SizeMode	StretchImage

pictureBox4 Picture Box:

Property Name	Property Value
Name	picHidden3
SizeMode	StretchImage

pictureBox5 Picture Box:

Property Name	Property Value
Name	picHidden4
SizeMode	StretchImage

pictureBox6 Picture Box:

Property Name	Property Value
Name	picHidden5
SizeMode	StretchImage

pictureBox7 Picture Box:

Property Name	Property Value
Name	picHidden6
SizeMode	StretchImage

pictureBox8 Picture Box:

Property Name	Property Value
Name	picHidden7
SizeMode	StretchImage

pictureBox9 Picture Box:

Property Name	Property Value
Name	picHidden8
SizeMode	StretchImage

pictureBox10 Picture Box:

Property Name	Property Value
Name	picHidden9
SizeMode	StretchImage

pictureBox11 Picture Box:

Property Name	Property Value
Name	picHidden10
SizeMode	StretchImage

pictureBox12 Picture Box:

Property Name	Property Value
Name	picHidden11
SizeMode	StretchImage

pictureBox13 Picture Box:

Property Name	Property Value
Name	picHidden12
SizeMode	StretchImage

pictureBox14 Picture Box:

Property Name	Property Value
Name	picHidden13
SizeMode	StretchImage

pictureBox15 Picture Box:

Property Name	Property Value
Name	picHidden14
SizeMode	StretchImage

pictureBox16 Picture Box:

Property Name	Property Value
Name	picHidden15
SizeMode	StretchImage

pictureBox17 Picture Box:

Property Name	Property Value
Name	picBack
SizeMode	StretchImage
Image	Back.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

pictureBox18 Picture Box:

Property Name	Property Value
Name	picChoice0
SizeMode	StretchImage
Image	Ball.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

pictureBox19 Picture Box:

Property Name	Property Value
Name	picChoice1
SizeMode	StretchImage
Image	Gift.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

pictureBox20 Picture Box:

Property Name	Property Value
Name	picChoice2
SizeMode	StretchImage
Image	Bear.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

pictureBox21 Picture Box:

Property Name	Property Value
Name	picChoice3
SizeMode	StretchImage
Image	Block.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

pictureBox22 Picture Box:

Property Name	Property Value
Name	picChoice4
SizeMode	StretchImage
Image	Ducky.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

pictureBox23 Picture Box:

Property Name	Property Value
Name	picChoice5
SizeMode	StretchImage
Image	Burger.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

pictureBox24 Picture Box:

Property Name	Property Value
Name	picChoice6
SizeMode	StretchImage
Image	Hotdog.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

pictureBox25 Picture Box:

Property Name	Property Value
Name	picChoice7
SizeMode	StretchImage
Image	Cake.gif (in \BeginVCS\BVCS Projects folder)
Visible	False

button1 Button:

Property Name	Property Value
Name	btnNew
Text	New Game

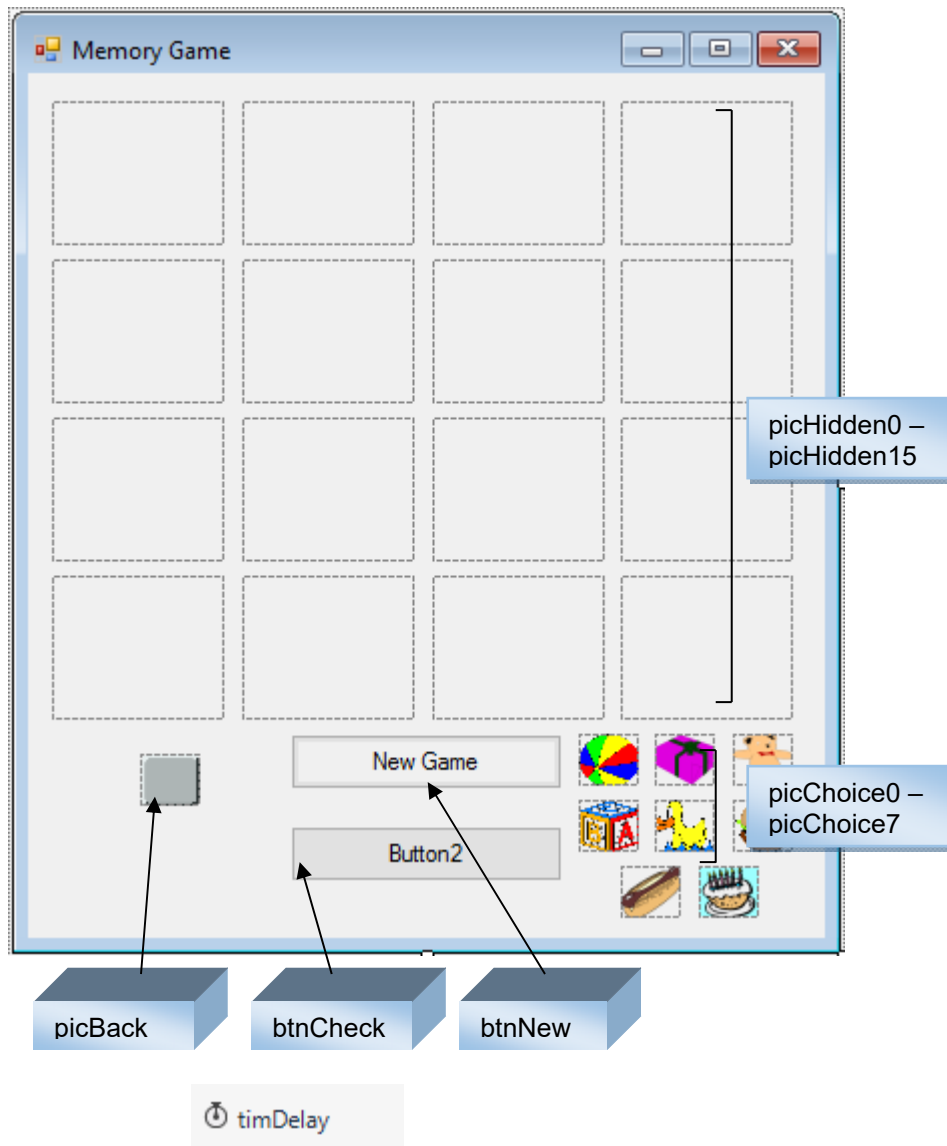
button2 Button:

Property Name	Property Value
Name	btnCheck

timer1 Timer:

Property Name	Property Value
Name	timDelay
Interval	1000

The completed form appears like this:



Before continuing, resize the form to hide **btnCheck**, a button we use for a general method. A few of the graphics will disappear, but that's okay since their `Visible` property is `False` anyway.

Write Event methods

When the game starts, pick one box, then another. The game stops when all matching picture pairs have been found. A delay is used to display the pictures for one second before deciding whether or not there is a match. At any time, click **New Game** to start again.

Add this code to the **general declarations** area:

```
int choice;  
int[] picked = new int[2];  
int[] behind = new int[16];  
PictureBox[] displayed = new PictureBox[16];  
PictureBox[] choices = new PictureBox[8];  
Random myRandom = new Random();
```

The **Form1_Load** event establishes images to pick from

```
private void Form1_Load(object sender, EventArgs e)
{
    //establish display and choices picture boxes
    displayed[0] = picHidden0;
    displayed[1] = picHidden1;
    displayed[2] = picHidden2;
    displayed[3] = picHidden3;
    displayed[4] = picHidden4;
    displayed[5] = picHidden5;
    displayed[6] = picHidden6;
    displayed[7] = picHidden7;
    displayed[8] = picHidden8;
    displayed[9] = picHidden9;
    displayed[10] = picHidden10;
    displayed[11] = picHidden11;
    displayed[12] = picHidden12;
    displayed[13] = picHidden13;
    displayed[14] = picHidden14;
    displayed[15] = picHidden15;
    choices[0] = picChoice0;
    choices[1] = picChoice1;
    choices[2] = picChoice2;
    choices[3] = picChoice3;
    choices[4] = picChoice4;
    choices[5] = picChoice5;
    choices[6] = picChoice6;
    choices[7] = picChoice7;
    // start new game
    btnNew.PerformClick();
}
```

The `btnNew_Click` event method sets up the hidden pictures:

```
private void btnNew_Click(object sender, EventArgs e)
{
    for (int i = 0; i < 16; i++)
    {
        // replace with card back
        displayed[i].Image = picBack.Image;
        displayed[i].Visible = true;
        behind[i] = i;
    }
    // Randomly sort 16 integers (0 to 15) using Shuffle
    routine from Class 9
    // behind array contains indexes (0-7) for hidden
    pictures
    // Work through remaining values
    // Start at 16 and swap one value
    // at each for loop step
    // After each step, remaining is decreased by 1
    for (int remaining = 16; remaining >= 1; remaining--)
    {
        // Pick item at random
        int itemPicked = myRandom.Next(remaining);
        // Swap picked item with bottom item
        int tempValue = behind[itemPicked];
        behind[itemPicked] = behind[remaining - 1];
        behind[remaining - 1] = tempValue;
    }
    for (int i = 0; i < 16; i++)
    {
        if (behind[i] > 7)
        {
            behind[i] = behind[i] - 8;
        }
    }
    choice = 0;
}
```

The **Click** event methods for the 16 picture boxes:

```
private void picHidden0_Click(object sender, EventArgs e)
{
    picked[choice] = 0;
    btnCheck.PerformClick();
}

private void picHidden1_Click(object sender, EventArgs e)
{
    picked[choice] = 1;
    btnCheck.PerformClick();
}

private void picHidden2_Click(object sender, EventArgs e)
{
    picked[choice] = 2;
    btnCheck.PerformClick();
}

private void picHidden3_Click(object sender, EventArgs e)
{
    picked[choice] = 3;
    btnCheck.PerformClick();
}

private void picHidden4_Click(object sender, EventArgs e)
{
    picked[choice] = 4;
    btnCheck.PerformClick();
}

private void picHidden5_Click(object sender, EventArgs e)
{
    picked[choice] = 5;
    btnCheck.PerformClick();
}

private void picHidden6_Click(object sender, EventArgs e)
{
    picked[choice] = 6;
    btnCheck.PerformClick();
}
```



```
private void picHidden7_Click(object sender, EventArgs e)
{
    picked[choice] = 7;
    btnCheck.PerformClick();
}

private void picHidden8_Click(object sender, EventArgs e)
{
    picked[choice] = 8;
    btnCheck.PerformClick();
}

private void picHidden9_Click(object sender, EventArgs e)
{
    picked[choice] = 9;
    btnCheck.PerformClick();
}

private void picHidden10_Click(object sender, EventArgs e)
{
    picked[choice] = 10;
    btnCheck.PerformClick();
}

private void picHidden11_Click(object sender, EventArgs e)
{
    picked[choice] = 11;
    btnCheck.PerformClick();
}

private void picHidden12_Click(object sender, EventArgs e)
{
    picked[choice] = 12;
    btnCheck.PerformClick();
}

private void picHidden13_Click(object sender, EventArgs e)
{
    picked[choice] = 13;
    btnCheck.PerformClick();
}
```

```
private void picHidden14_Click(object sender, EventArgs e)
{
    picked[choice] = 14;
    btnCheck.PerformClick();
}

private void picHidden15_Click(object sender, EventArgs e)
{
    picked[choice] = 15;
    btnCheck.PerformClick();
}
```

The **btnCheck_Click** “hidden” general method that displays the choices for a match:

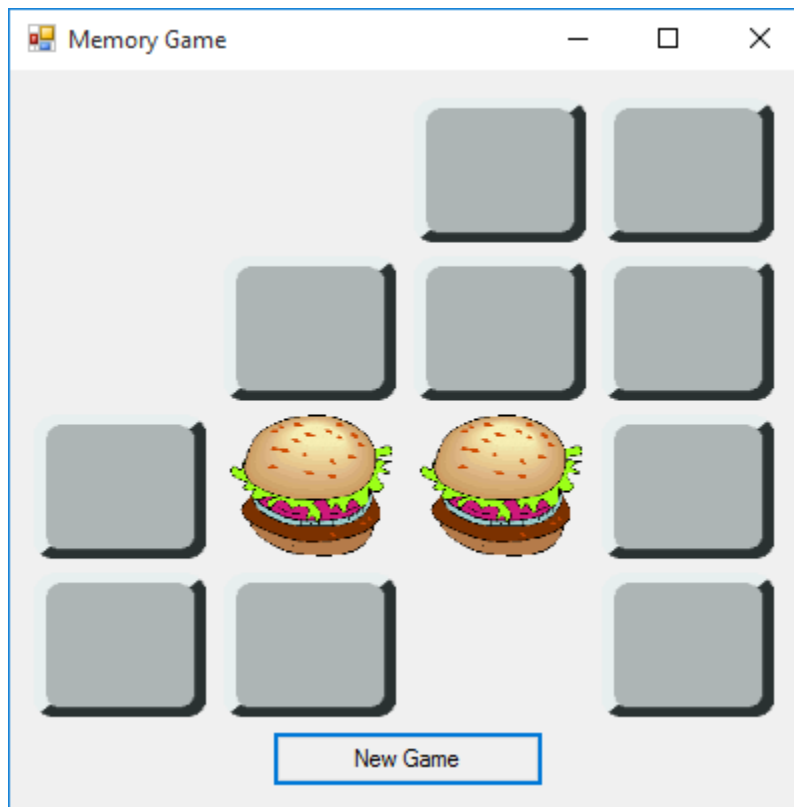
```
private void btnCheck_Click(object sender, EventArgs e)
{
    // Only execute if not trying to pick same box
    if (choice == 0 || (choice == 1 && picked[0] !=
picked[1]))
    {
        // Display selected picture
        displayed[picked[choice]].Image =
choices[behind[picked[choice]]].Image;
        displayed[picked[choice]].Refresh();
        if (choice == 0)
        {
            // first choice - just display
            choice = 1;
        }
        else
        {
            // Delay for one second before checking
            timDelay.Enabled = true;
        }
    }
}
```

The `timDelay_Tick` method that checks for matches after a delay:

```
private void timDelay_Tick(object sender, EventArgs e)
{
    timDelay.Enabled = false;
    // After delay, check for match
    if (behind[picked[0]] == behind[picked[1]])
    {
        // If match, remove pictures
        displayed[picked[0]].Visible = false;
        displayed[picked[1]].Visible = false;
    }
    else
    {
        // If no match, blank picture, restore backs
        displayed[picked[0]].Image = picBack.Image;
        displayed[picked[1]].Image = picBack.Image;
    }
    choice = 0;
}
```

Run the Project

Save your work. Run the project. Sixteen boxes appear. Click on one and view the picture. Click on another. If there is a match, the two pictures are removed (after a delay). If there is no match, the boxes are restored (also after a delay). Once all matches are found, click **New Game** to play again. Here's the middle of a game I was playing (notice the form has been resized at design time to hide the lower of the two button controls):



Other Things to Try

Some things to help improve or change this game: add a scoring system to keep track of how many tries you took to find all the matches, make it a two player game where you compete against another player or the computer, or set it up to match other items (shapes, colors, upper and lower case letters, numbers and objects, etc.).

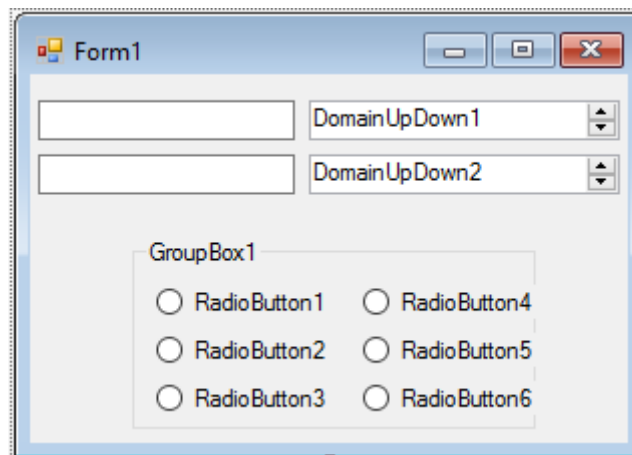
Project 6 – Units Conversion

Project Design

In this project, we will build a program that converts length from one unit of measure (inch, foot, yard, mile, centimeter, meter, kilometer) to another. The program will allow you to choose (using radio buttons) how many decimal points you want to display in the result. The project you are about to build is saved as **Conversion** in the project folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Place two text box controls and two domain updown controls on the form. The domain updown controls are like a numeric updown, with items listed instead of numbers. Also, place a group box with six radio buttons. When done, your form should look something like this:



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Units Conversion
BorderStyle	FixedSingle
StartPosition	CenterScreen

textBox1 Text Box:

Property Name	Property Value
Name	txtFromValue
Text	0
TextAlign	Right
Font Size	12

textBox2 Text Box:

Property Name	Property Value
Name	txtToValue
Text	0
TextAlign	Right
BackColor	White
Font Size	12
ReadOnly	True
TabStop	False

domainUpDown1 Domain UpDown:

Property Name	Property Value
Name	dudFromUnit
Text	[Blank]
TextAlign	Right
BackColor	Light Yellow
Font Size	12
ReadOnly	True

domainUpDown2 Domain UpDown:

Property Name	Property Value
Name	dudToUnit
Text	[Blank]
TextAlign	Right
BackColor	Light Yellow
Font Size	12
ReadOnly	True

groupBox1 Group Box:

Property Name	Property Value
Name	grpConvert
Text	Number of Decimals
Font Size	10

radioButton1 Radio Button:

Property Name	Property Value
Name	rdoDec0
Text	0
Checked	True

radioButton2 Radio Button:

Property Name	Property Value
Name	rdoDec1
Text	1

radioButton3 Radio Button:

Property Name	Property Value
Name	rdoDec2
Text	2

radioButton4 Radio Button:

Property Name	Property Value
Name	rdoDec3
Text	3

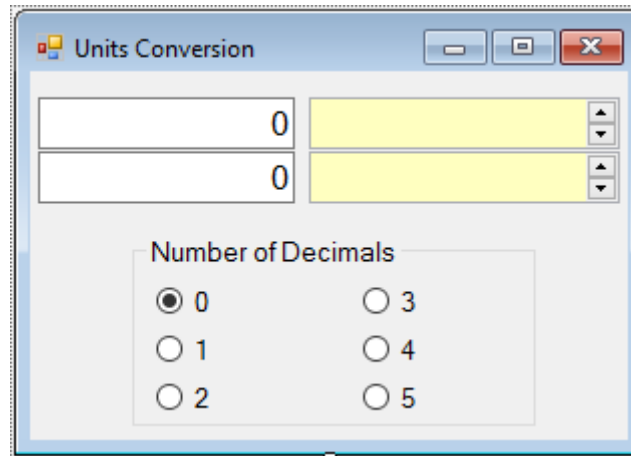
radioButton5 Radio Button:

Property Name	Property Value
Name	rdoDec4
Text	4

radioButton6 Radio Button:

Property Name	Property Value
Name	rdoDec5
Text	5

My finished form looks like this (resized some controls):



The screenshot shows a Windows application window titled "Units Conversion". Inside the window, there are two text boxes, each containing the number "0". To the right of each text box is a yellow dropdown menu. Below these controls is a group box titled "Number of Decimals". Inside this group box, there are six radio buttons arranged in two columns. The first column has radio buttons for "0", "1", and "2", with the "0" button selected. The second column has radio buttons for "3", "4", and "5".

Write Event Methods

The idea of the program is simple. Type a value in the text box and choose the units (both From and To). With each change in value, units or number of decimals, the conversion is updated. Most of the computation is involved with the **txtFromValue_Change** event. Note how the conversion factors are stored in a two-dimensional array (table). Also note the use of the **NumberFormatInfo** object to specify the number of decimals to display. To use this object, you need these two lines after the left brace declaring the project **namespace** (right at the top of the code window):

```
using System;
using System.Globalization;
```

Add this code to the **general declarations** area:

```
string[] units = new string[7];
double [,] conversions = new double[7, 7];
bool loading = true;
NumberFormatInfo provider = new CultureInfo("en-US",
false).NumberFormat;
```

The **Form1_Load** event method:

```
private void Form1_Load(object sender, EventArgs e)
{
    int i;
    // Establish conversion factors - stored in two
dimensional array
    // or table - the first number is the table row, the
second number
    // the table column
    conversions[0, 0] = 1; //in to in
    conversions[0, 1] = 1.0 / 12; //in to ft
    conversions[0, 2] = 1.0 / 36; // in to yd
```

```
conversions[0, 3] = (1.0 / 12) / 5280; // in to mi
conversions[0, 4] = 2.54; // in to cm
conversions[0, 5] = 2.54 / 100; // in to m
conversions[0, 6] = 2.54 / 100000; // in to km
for (i = 0; i < 7; i++)
{
    conversions[1, i] = 12 * conversions[0, i];
    conversions[2, i] = 36 * conversions[0, i];
    conversions[3, i] = 5280 * (12 * conversions[0, i]);
    conversions[4, i] = conversions[0, i] / 2.54;
    conversions[5, i] = 100 * conversions[0, i] / 2.54;
    conversions[6, i] = 100000 * (conversions[0, i] /
2.54);
}
// Initialize variables
units[0] = "inches (in)";
units[1] = "feet (ft)";
units[2] = "yards (yd)";
units[3] = "miles (mi)";
units[4] = "centimeters (cm)";
units[5] = "meters (m)";
units[6] = "kilometers (km)";
for (i = 0; i < 7; i++)
{
    dudFromUnit.Items.Add(units[i]);
    dudToUnit.Items.Add(units[i]);
}
dudFromUnit.SelectedIndex = 0;
dudToUnit.SelectedIndex = 0;
provider.NumberDecimalDigits = 0;
// Put cursor in text box
txtFromValue.Focus();
loading = false;
}
```

The `txtFromValue_KeyPress` event method:

```
private void txtFromValue_KeyPress(object sender,
KeyPressEventArgs e)
{
    // Numbers and decimal point only
    if ((e.KeyChar >= '0' && e.KeyChar <= '9') || e.KeyChar
== '.' || (int) e.KeyChar == 8)
    {
        e.Handled = false;
    }
    else
    {
        e.Handled = true;
    }
}
```

The `txtFromValue_TextChanged` event method:

```
private void txtFromValue_TextChanged(object sender,
EventArgs e)
{
    UpdateDisplay();
}
```

The **UpdateDisplay** general method (recall type the method after any other method):

```
private void UpdateDisplay()
{
    if (loading)
    {
        return;
    }
    double v;
    // Do unit conversion
    v = conversions[dudFromUnit.SelectedIndex,
dudToUnit.SelectedIndex] *
Convert.ToDouble(txtFromValue.Text);
    txtToValue.Text = v.ToString("N", provider);
    txtFromValue.Focus();
}
```

The **dudFromUnit_SelectedItemChanged** event method:

```
private void dudFromUnit_SelectedItemChanged(object sender,
EventArgs e)
{
    UpdateDisplay();
}
```

The **dudToUnit_SelectedItemChanged** event method:

```
private void dudToUnit_SelectedItemChanged(object sender,
EventArgs e)
{
    UpdateDisplay();
}
```

The `rdoDec0_CheckedChanged` to `rdoDec5_CheckedChanged` event methods:

```
private void rdoDec0_CheckedChanged(object sender, EventArgs
e)
{
    provider.NumberDecimalDigits = 0;
    UpdateDisplay();
}

private void rdoDec1_CheckedChanged(object sender, EventArgs
e)
{
    provider.NumberDecimalDigits = 1;
    UpdateDisplay();
}

private void rdoDec2_CheckedChanged(object sender, EventArgs
e)
{
    provider.NumberDecimalDigits = 2;
    UpdateDisplay();
}

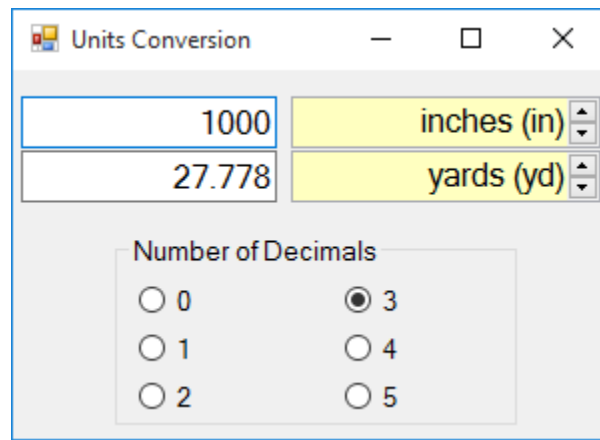
private void rdoDec3_CheckedChanged(object sender, EventArgs
e)
{
    provider.NumberDecimalDigits = 3;
    UpdateDisplay();
}

private void rdoDec4_CheckedChanged(object sender, EventArgs
e)
{
    provider.NumberDecimalDigits = 4;
    UpdateDisplay();
}

private void rdoDec5_CheckedChanged(object sender, EventArgs
e)
{
    provider.NumberDecimalDigits = 5;
    UpdateDisplay();
}
```

Run the Project

Save your work. Run the project. Type in a value. Watch the corresponding converted value change as you type. Change the From units and the To units using the updown controls. Change the number of decimals. Make sure all the options work as designed. Here's a run I tried:



Other Things to Try

The most obvious change to this program is to include other units of measure. You could build a general purpose units conversion program that converts not only length, but weight, volume, density, area, temperature and many others. Such a program would be invaluable.

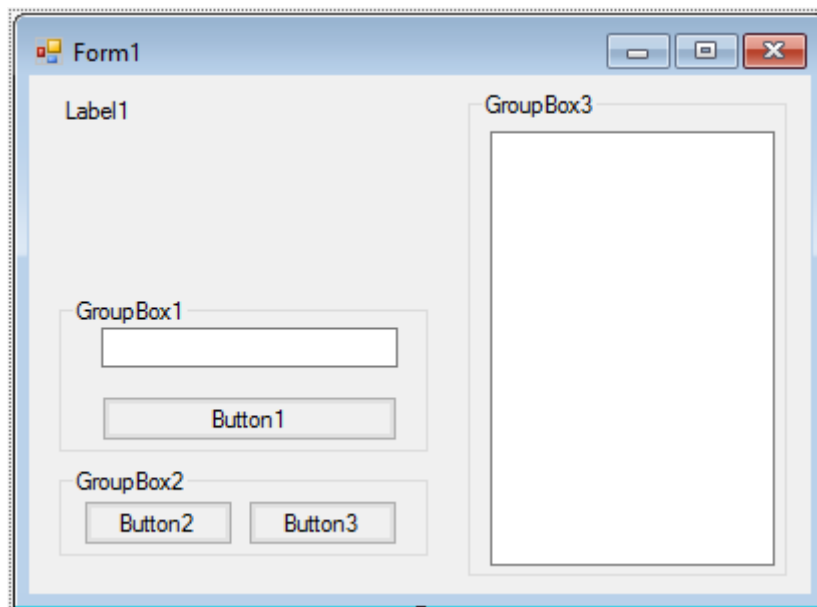
Project 7 - Decode

Project Design

This project is a classic computer game. The computer generates a four-digit code (with no repeating digits). You guess at the code. The computer then tells you how many digits in your guess are correct and how many digits are in the correct location. Based on these clues, you make a new guess. You continue guessing until you have cracked the code. The project you are about to build is saved as **Decode** in the project folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Place a label control (set **AutoSize** to **False** so it can be resized), two button controls and two group box controls on the form. In the first group box, place a text box control and a button. In the second group box, place a text box control. When done, your form should look something like this:



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Decode
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

label1 Label:

Property Name	Property Value
Name	lblMessage
Text	[Blank]
TextAlign	MiddleCenter
BorderStyle	Fixed3D
BackColor	White
ForeColor	Blue
Font Size	12

groupBox1 Group Box:

Property Name	Property Value
Name	grpGuess
BackColor	Red
Text	[Blank]
Visible	False

textBox1 Text Box:

Property Name	Property Value
Name	txtGuess
TextAlign	Center
Font	Courier New
Font Size	16
MaxLength	4

button1 Button:

Property Name	Property Value
Name	btnCheck
Text	Check Guess
Font	Arial
Font Size	12
BackColor	Light Yellow

button2 Button:

Property Name	Property Value
Name	btnNew
Text	New Game

button3 Button:

Property Name	Property Value
Name	btnStop
Text	Stop
Enabled	False

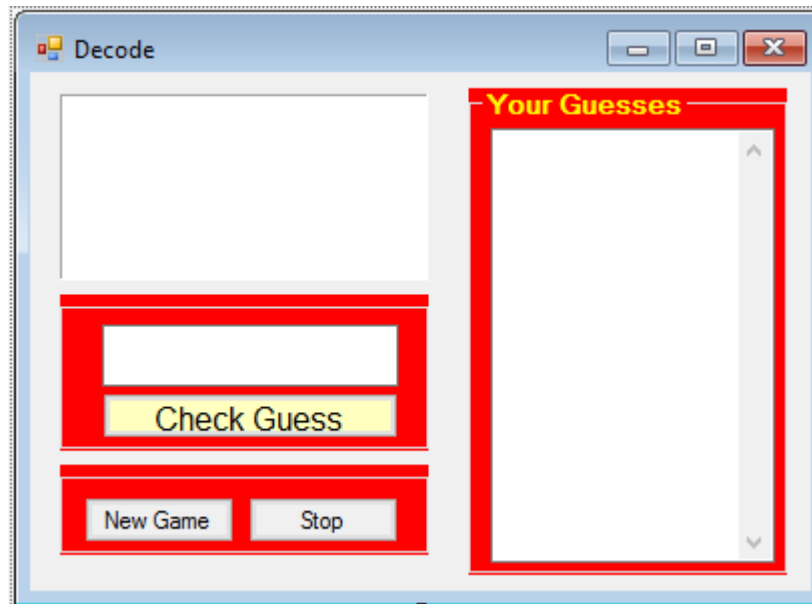
groupBox2 Group Box:

Property Name	Property Value
Name	grpGuesses
Text	Your Guesses
BackColor	Red
ForeColor	Yellow
Font Size	10
Font Style	Bold

textBox2 Text Box:

Property Name	Property Value
Name	txtGuesses
Font	Courier New
Font Size	14
BackColor	White
MultiLine	True
ReadOnly	True
TabStop	False
ScrollBars	Vertical

When done setting properties, my form looks like this:



Write Event Methods

Most of the code in this project is involved with generating a four-digit computer code (when you click **New Game**) and checking the guess you input (click **Check Guess**). The **Your Guesses** group box provides a history of each guess you made.

Add this code to the **general declarations** area:

```
bool gameOver;  
string computerCode;  
string[] computerNumbers = new string[4];  
Random myRandom = new Random();
```

The **Form1_Load** event method:

```
private void Form1_Load(object sender, EventArgs e)  
{  
    lblMessage.Text = "Click New Game";  
    btnNew.Focus();  
}
```

The `btnNew_Click` event method:

```
private void btnNew_Click(object sender, EventArgs e)
{
    string[] nArray = new string[10];
    int i, j;
    string t;
    // Start new game
    btnStop.Enabled = true;
    gameOver = false;
    lblMessage.Text = "";
    txtGuesses.Text = "";
    txtGuess.Text = "";
    btnNew.Enabled = false;
    // Choose code using modified version of card shuffling
    routine
    // Order all digits initially
    computerCode = "";
    for (i = 0; i < 10; i++)
    {
        nArray[i] = Convert.ToString(i);
    }
    // J is number of integers remaining
    for (j = 9; j >= 6; j--)
    {
        i = myRandom.Next(j);
        computerNumbers[9 - j] = nArray[i];
        computerCode = computerCode + nArray[i];
        t = nArray[j];
        nArray[j] = nArray[i];
        nArray[i] = t;
    }
    lblMessage.Text = "I have a 4 digit code.\r\nTry to guess
it.";
    grpGuess.Visible = true;
    txtGuess.Focus();
}
```

The **btnStop_Click** event method:

```
private void btnStop_Click(object sender, EventArgs e)
{
    // Stop current game
    grpGuess.Visible = false;
    btnNew.Enabled = true;
    btnStop.Enabled = false;
    if (!gameOver)
    {
        lblMessage.Text = "Game Stopped\r\nMy code was - " +
computerCode;
    }
    btnNew.Focus();
}
```

The **txtGuess_KeyPress** event method:

```
private void txtGuess_KeyPress(object sender,
KeyPressEventArgs e)
{
    // Allow numbers only
    if ((int) e.KeyChar == 13)
    {
        btnCheck.PerformClick();
    }
    else if ((e.KeyChar >= '0' && e.KeyChar <= '9') || (int)
e.KeyChar == 8)
    {
        e.Handled = false;
    }
    else
    {
        e.Handled = true;
    }
}
```

The `btnCheck_Click` event method:

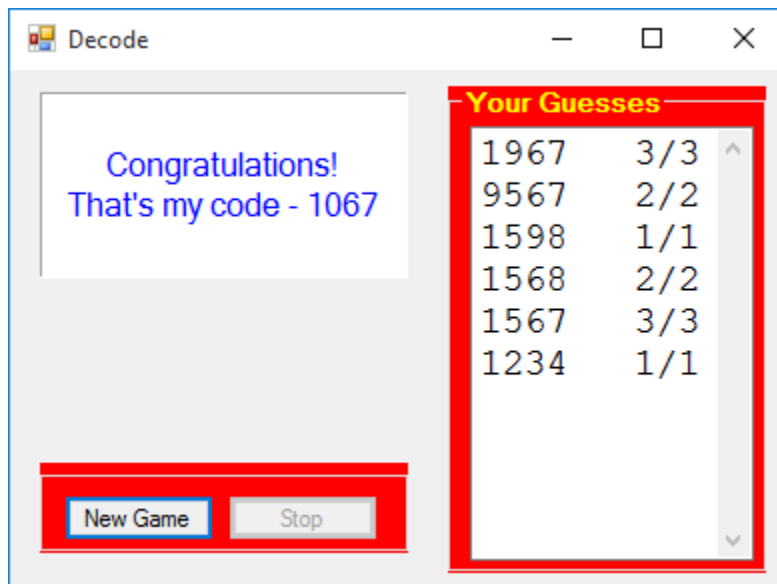
```
private void btnCheck_Click(object sender, EventArgs e)
{
    string w;
    string[] wNumbers = new string[4];
    int i, j, k1, k2;
    bool distinct;
    // Check your guess
    w = txtGuess.Text;
    // Check length validity
    if (w.Length != 4)
    {
        lblMessage.Text = "Guess must have 4 numbers
...\\r\\nTry again!";
        txtGuess.Focus();
        return;
    }
    else
    {
        // Get numbers and make sure they are distinct
        distinct = true;
        for (i = 0; i < 4; i++)
        {
            wNumbers[i] = w.Substring(i, 1);
            if (i != 0)
            {
                for (j = i - 1; j >= 0 ; j--)
                {
                    if (wNumbers[i] == wNumbers[j])
                    {
                        distinct = false;
                    }
                }
            }
        }
        if (!distinct)
        {
            lblMessage.Text = "Numbers must all be different
...\\r\\nTry again!";
            txtGuess.Focus();
            return;
        }
        if (w == computerCode)
        {

```

```
        lblMessage.Text = "Congratulations!\r\nThat's my
code - " + computerCode;
        gameOver = true;
        btnStop.PerformClick();
        return;
    }
    else
    {
        // Compute score
        k1 = 0;
        k2 = 0;
        for (i = 0; i < 4; i++)
        {
            for (j = 0; j < 4; j++)
            {
                if (wNumbers[j] == computerNumbers[i])
                    k1++;
            }
            if (wNumbers[i] == computerNumbers[i])
                k2++;
        }
        lblMessage.Text = "Your guess - " + w + "\r\n" +
Convert.ToString(k1) + " digit(s) correct\r\n" +
Convert.ToString(k2) + " digit(s) in proper place";
        txtGuesses.Text = w + " " +
Convert.ToString(k1) + "/" + Convert.ToString(k2) + "\r\n" +
txtGuesses.Text;
        txtGuess.Text = "";
        txtGuess.Focus();
    }
}
}
```

Run the Project

Save your work. Run the project. Click **New Game** to start. Type a guess for the four-digit code. Note the computer will not let you type an illegal guess (non-distinct, less than 4 digits). Click **Check Guess**. After each guess, the computer will tell you how many digits are correct and how many are in the correct location. For your reference, a history of your guesses is displayed under **Your Guesses**. The score is displayed as two numbers separated by a slash. The first number is the number of correct digits, the second the number in the correct location. Click **Stop** to stop guessing and see the computer's code. Here's a game I played:



Other Things to Try

You can give this game a variable difficulty by allowing the user to choose how many digits are in the code, how many numbers are used to generate the code, and whether digits can repeat. See if you can code up and implement some of these options. The commercial version of this game (called MasterMind) uses colored pegs to set the code. This makes for a prettier game. See if you can code this variation.

Lastly, many mathematical papers have been written on developing a computer program that can decode the kinds of codes used here. Do you think you could write a computer program to determine a four digit code you make up? The program would work like this: (1) computer makes a guess, (2) you tell computer how many digits are correct and how many are in correct locations, then, (3) computer generates a new guess. The computer would continue guessing until it gave up or guessed your code.

Project 8 - Frown

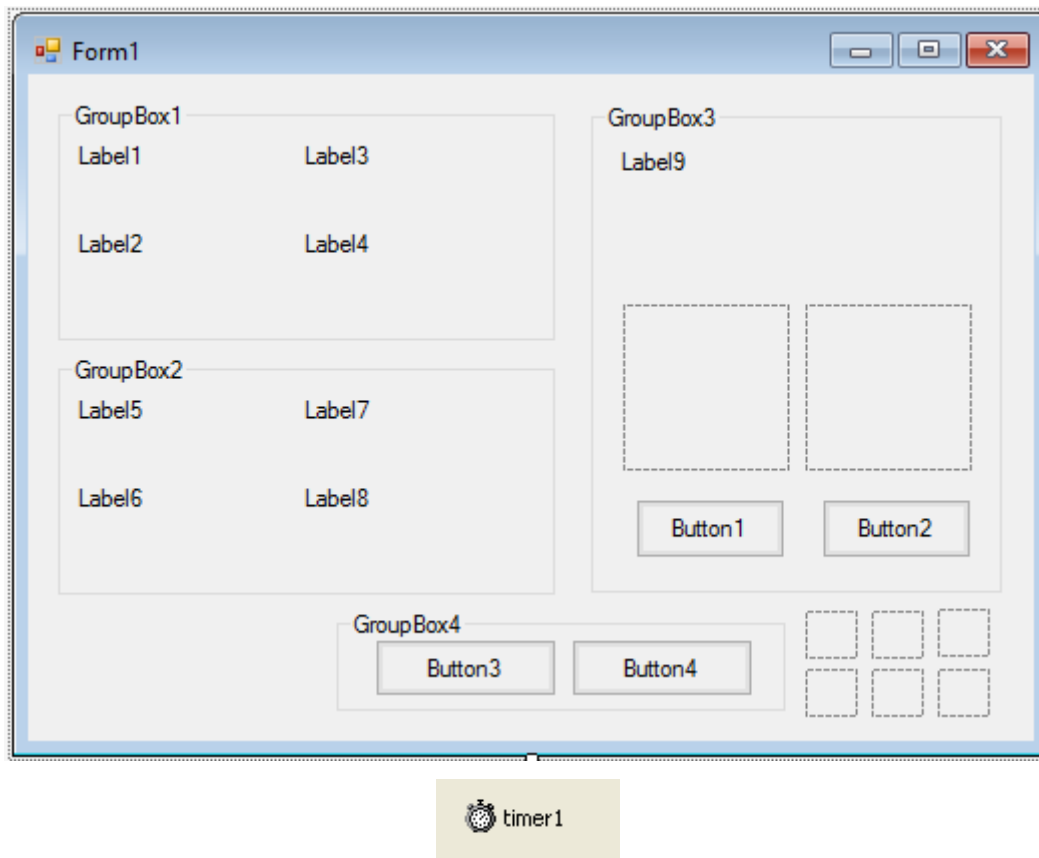
Project Design

Frown is a fun two-player dice game you play against the computer. You play with a set of two dice that are normal except the side of the die where the “1” would be is replaced by a frowning face. The object of the game is to achieve a score of 100 points. Players alternate turns, which consist of a series of at least one roll of the dice, perhaps many, subject to the following rules.

As long as no frown appears on either die, the roller builds a running score for the current turn. After each roll with no frown, the player can choose to continue rolling or pass the dice to the other player. If the player passes the dice, the current score is added to any previous total. If a frown appears, the player loses the points gained on the current turn. If two frowns appear, the player loses the current points and all saved points! There is a considerable amount of luck involved. However, the skill of deciding when to pass the dice to your opponent also figures prominently. The project you are about to build is saved as **Frown** in the project folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. There are lots of controls here. Place two group box controls on the form. Place four label controls in each of the group boxes. Put two buttons below the second group box. Add a panel control – in this panel, place a label, two picture box controls, and two buttons. Finally, add a timer control and six small picture box controls to the form. Set **AutoSize** to **False** for each label to allow resizing. When done, your form should look something like this:



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Frown
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

groupBox1 Group Box:

Property Name	Property Value
Name	grpYou
Text	You
BackColor	Blue
ForeColor	Yellow
Font Size	12
Font Style	Bold

label1 Label:

Property Name	Property Value
Text	Score This Turn
ForeColor	White
Font Size	10

label2 Label:

Property Name	Property Value
Text	Total Score
ForeColor	White
Font Size	10

label3 Label:

Property Name	Property Value
Name	lblYouScore
AutoSize	False
Text	[Blank]
TextAlign	MiddleCenter
BorderStyle	Fixed3D
BackColor	White
ForeColor	Black
Font Size	12

label4 Label:

Property Name	Property Value
Name	lblYouTotal
AutoSize	False
Text	0
TextAlign	MiddleCenter
BorderStyle	Fixed3D
BackColor	White
ForeColor	Black
Font Size	12

groupBox2 Group Box:

Property Name	Property Value
Name	grpComputer
Text	Computer
BackColor	Blue
ForeColor	Yellow
Font Size	12
Font Style	Bold

label5 Label:

Property Name	Property Value
Text	Score This Turn
TextAlign	MiddleLeft
ForeColor	White
Font Size	10

label6 Label:

Property Name	Property Value
Text	Total Score
TextAlign	MiddleLeft
ForeColor	White
Font Size	10

label7 Label:

Property Name	Property Value
Name	lblComputerScore
AutoSize	False
Text	[Blank]
TextAlign	MiddleCenter
BorderStyle	Fixed3D
BackColor	White
ForeColor	Black
Font Size	12

label8 Label:

Property Name	Property Value
Name	lblComputerTotal
AutoSize	False
Text	0
TextAlign	MiddleCenter
BorderStyle	Fixed3D
BackColor	White
ForeColor	Black
Font Size	12

panel1 Panel:

Property Name	Property Value
Name	pnlDice
BackColor	Red

label9 Label:

Property Name	Property Value
Name	lblMessage
Text	[Blank]
TextAlign	MiddleCenter
BorderStyle	Fixed3D
BackColor	Light Yellow
Font Size	10

pictureBox1 Picture Box:

Property Name	Property Value
Name	picDice1
BackColor	Green
SizeMode	StretchImage

pictureBox2 Picture Box:

Property Name	Property Value
Name	picDice2
BackColor	Green
SizeMode	StretchImage

button1 Button:

Property Name	Property Value
Name	btnRoll
Text	Roll Dice
BackColor	Light Red
Enabled	False

button2 Button:

Property Name	Property Value
Name	btnPass
Text	Pass Dice
BackColor	Light Red
Enabled	False

button3 Button:

Property Name	Property Value
Name	btnNew
Text	New Game

button4 Button:

Property Name	Property Value
Name	btnStop
Text	Exit

timer1 Timer:

Property Name	Property Value
Name	timComputer
Interval	2000

pictureBox3 PictureBox:

Property Name	Property Value
Name	picDots1
Image	frown.gif (in the \BeginVCS\BVCS Projects\Frown folder)
SizeMode	StretchImage
Visible	False

pictureBox4 PictureBox:

Property Name	Property Value
Name	picDots2
Image	dice2.gif (in the \BeginVCS\BVCS Projects\Frown folder)
SizeMode	StretchImage
Visible	False

pictureBox5 PictureBox:

Property Name	Property Value
Name	picDots3
Image	dice3.gif (in the \BeginVCS\BVCS Projects\Frown folder)
SizeMode	StretchImage
Visible	False

pictureBox6 PictureBox:

Property Name	Property Value
Name	picDots4
Image	dice4.gif (in the \BeginVCS\BVCS Projects\Frown folder)
SizeMode	StretchImage
Visible	False

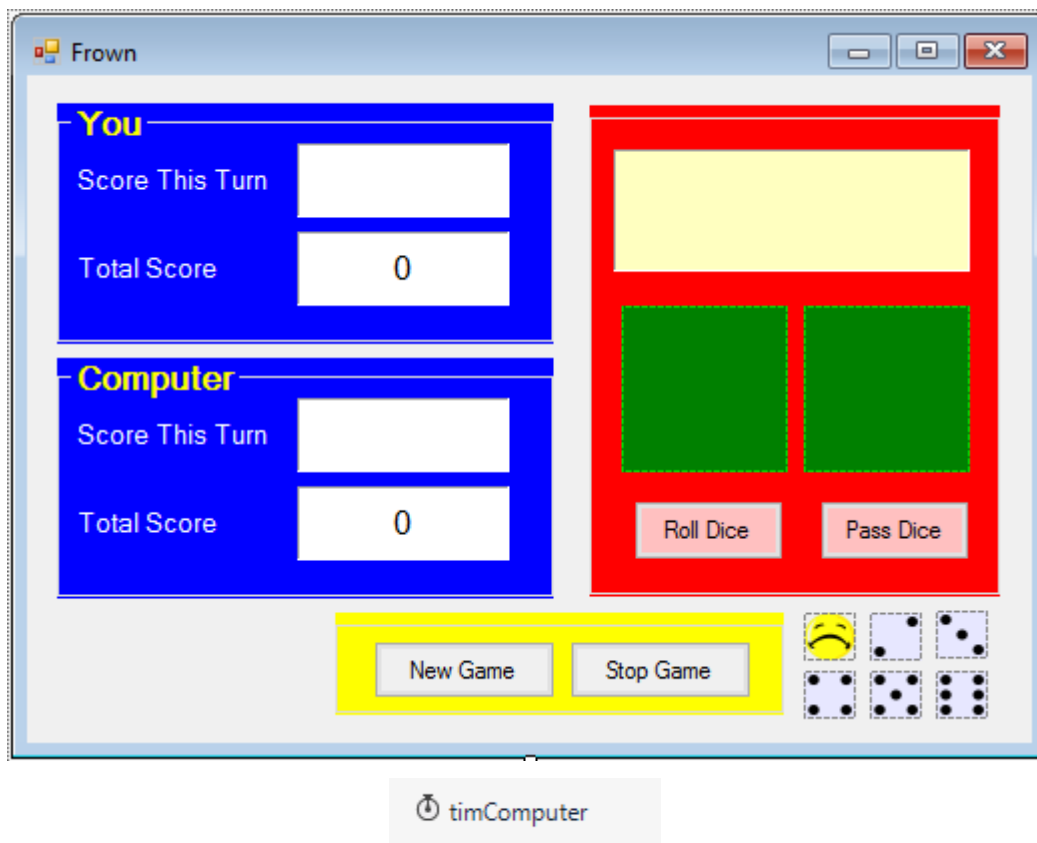
pictureBox7 PictureBox:

Property Name	Property Value
Name	picDots5
Image	dice5.gif (in the \BeginVCS\BVCS Projects\Frown folder)
SizeMode	StretchImage
Visible	False

pictureBox8 PictureBox:

Property Name	Property Value
Name	picDots6
Image	dice6.gif (in the \BeginVCS\BVCS Projects\Frown folder)
SizeMode	StretchImage
Visible	False

When done setting properties, my form looks like this:



Write Event Methods

Most of the code is involved with randomly rolling the two dice and passing control of the game from one player to the other. Study the logic carefully – it is used in many games where the human plays against the computer.

Add this code to the **general declarations** area:

```
bool gameOver;  
int whoseTurn, dice1, dice2;  
int youScore, computerScore;  
int youTotal, computerTotal;  
const int win = 100;  
Random myRandom = new Random();
```

The **Form1_Load** event method:

```
private void Form1_Load(object sender, EventArgs e)  
{  
    // Initialize dice to frowns  
    lblMessage.Text = "Click New Game To Start";  
    picDice1.Image = picDots1.Image;  
    picDice2.Image = picDots1.Image;  
    btnNew.Focus();  
}
```

The `btnNew_Click` event method:

```
private void btnNew_Click(object sender, EventArgs e)
{
    // Start new game
    gameOver = false;
    lblMessage.Text = "";
    btnNew.Enabled = false;
    btnStop.Text = "Stop Game";
    youScore = 0;
    lblYouScore.Text = "";
    computerScore = 0;
    lblComputerScore.Text = "";
    youTotal = 0;
    lblYouTotal.Text = "0";
    computerTotal = 0;
    lblComputerTotal.Text = "0";
    if (myRandom.Next(2) == 0)
    {
        // Computer goes first
        whoseTurn = 0;
        lblComputerScore.Text = "0";
        lblMessage.Text = "I'll roll first.";
        // must call instead of performclick since button is
        // is not enabled
        btnRoll_Click(null, null);
    }
    else
    {
        // You go first
        whoseTurn = 1;
        lblYouScore.Text = "0";
        lblMessage.Text = "You roll first.";
        btnRoll.Enabled = true;
        btnRoll.Focus();
    }
}
```

The **btnStop_Click** event method:

```
private void btnStop_Click(object sender, EventArgs e)
{
    if (btnStop.Text == "Exit")
    {
        this.Close();
    }
    else
    {
        // Stop current game
        timComputer.Enabled = false;
        btnNew.Enabled = true;
        btnStop.Text = "Exit";
        btnRoll.Enabled = false;
        btnPass.Enabled = false;
        if (!gameOver)
        {
            lblMessage.Text = "Game Stopped";
        }
        btnNew.Focus();
    }
}
```

The **btnRoll_Click** event method:

```
private void btnRoll_Click(object sender, EventArgs e)
{
    // Dice rolling
    // Roll Dice 1 and set display
    dice1 = myRandom.Next(6) + 1;
    switch (dice1)
    {
        case 1:
            picDice1.Image = picDots1.Image;
            break;
        case 2:
            picDice1.Image = picDots2.Image;
            break;
        case 3:
            picDice1.Image = picDots3.Image;
            break;
        case 4:
            picDice1.Image = picDots4.Image;
```

```

        break;
    case 5:
        picDice1.Image = picDots5.Image;
        break;
    case 6:
        picDice1.Image = picDots6.Image;
        break;
}
// Roll Dice 2 and set display
dice2 = myRandom.Next(6) + 1;
switch (dice2)
{
    case 1:
        picDice2.Image = picDots1.Image;
        break;
    case 2:
        picDice2.Image = picDots2.Image;
        break;
    case 3:
        picDice2.Image = picDots3.Image;
        break;
    case 4:
        picDice2.Image = picDots4.Image;
        break;
    case 5:
        picDice2.Image = picDots5.Image;
        break;
    case 6:
        picDice2.Image = picDots6.Image;
        break;
}
picDice1.Refresh();
picDice2.Refresh();
if (whoseTurn == 0)
{
    // Computer rolled
    if (dice1 > 1 && dice2 > 1)
    {
        // No frowns
        computerScore = computerScore + dice1 + dice2;
        lblComputerScore.Text =
Convert.ToString(computerScore);
        timComputer.Enabled = true;
        lblMessage.Text = lblMessage.Text + " Let me
think ...";
        return;
    }
}

```

```
        else if (dice1 == 1 && dice2 == 1)
        {
            // Two frowns - lose everything - must pass
            lblMessage.Text = lblMessage.Text + "\r\nI lost
all my points!\r\nYour turn.";
            computerTotal = 0;
            lblComputerTotal.Text = "0";
        }
        else
        {
            // One frown - must pass
            lblMessage.Text = lblMessage.Text + "\r\nI lost
my turn.\r\nYour turn.";
        }
        computerScore = 0;
        lblComputerScore.Text = "";
        whoseTurn = 1;
        btnRoll.Enabled = true;
        btnRoll.Focus();
    }
    else
    {
        // You rolled
        lblMessage.Text = "Still your turn.";
        btnPass.Enabled = true;
        if (dice1 > 1 && dice2 > 1)
        {
            // No frowns
            youScore = youScore + dice1 + dice2;
            lblYouScore.Text = Convert.ToString(youScore);
        }
        else if (dice1 == 1 && dice2 == 1)
        {
            // Two frowns - lose everything - must pass
            youScore = 0;
            youTotal = 0;
            lblMessage.Text = "You lost everything.\r\nYou
must pass to me.";
            btnRoll.Enabled = false;
            btnPass.Focus();
        }
    }
}
```

```

        else
        {
            // One frown - must pass
            youScore = 0;
            lblMessage.Text = "You lost your turn.\r\nYou
must pass to me.";
            btnRoll.Enabled = false;
            btnPass.Focus();
        }
    }
}

```

The **btnPass_Click** event method:

```

private void btnPass_Click(object sender, EventArgs e)
{
    // You passed dice to computer
    btnRoll.Enabled = false;
    btnPass.Enabled = false;
    whoseTurn = 0;
    youTotal = youTotal + youScore;
    youScore = 0;
    lblYouScore.Text = "";
    lblYouTotal.Text = Convert.ToString(youTotal);
    if (youTotal >= win)
    {
        gameOver = true;
        lblMessage.Text = "You win!!";
        btnStop.PerformClick();
    }
    else
    {
        lblMessage.Text = "I'll roll now.";
        // call btnroll routine, we can't use performclick
        // method since button is not enabled at this point
        btnRoll_Click(null, null);
    }
}

```

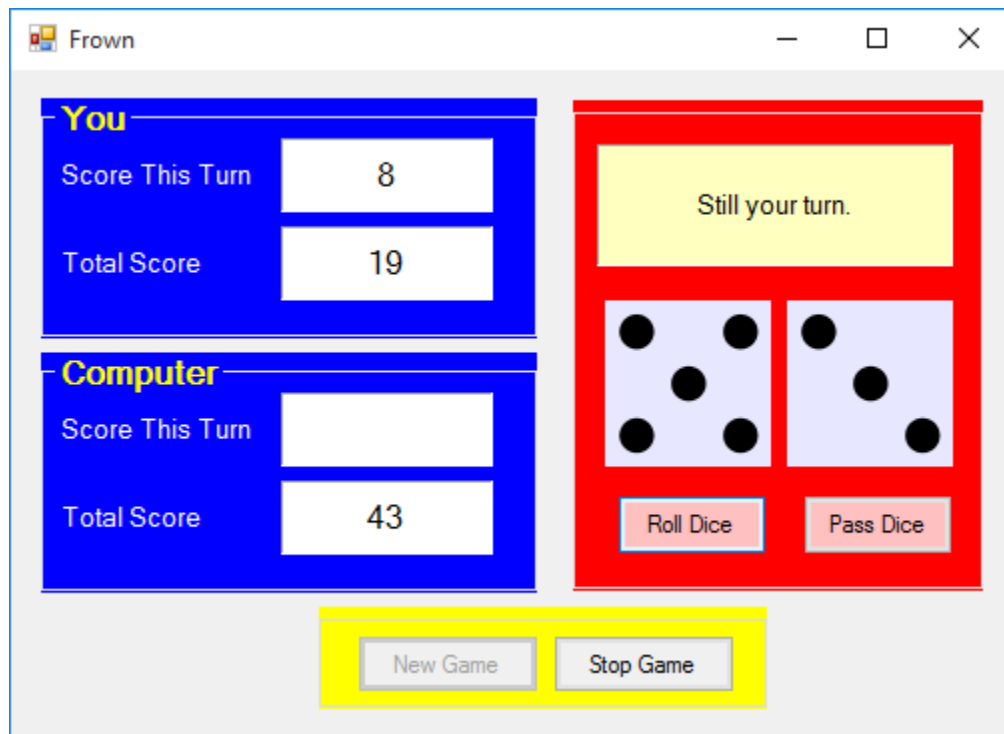
The `timComputer_Timer` event method:

```
private void timComputer_Tick(object sender, EventArgs e)
{
    int v;
    int odds;
    // Computer turn - decide wheter to roll again or pass
    timComputer.Enabled = false;
    v = computerScore + computerTotal;
    if (v >= win)
    {
        // Computer wins!
        gameOver = true;
        lblComputerTotal.Text = Convert.ToString(v);
        lblMessage.Text = "I win!!";
        btnStop.PerformClick();
        return;
    }
    else if (win - youTotal <= 10)
    {
        // If you are close to win, computer rolls again
        lblMessage.Text = "I'll roll again.";
        btnRoll_Click(null, null);
    }
    else
    {
        if (computerTotal >= youTotal)
        {
            // If computer already ahead, less likely to roll
again
            odds = (int) (100 * ((double) computerScore) /
30.0);
        }
        else if (v < youTotal)
        {
            // If computer behind, more likely
50.0);
        }
        else
        {
            odds = (int) (100.0 * ((double)computerScore) /
40.0);
        }
        if (myRandom.Next(100) > odds)
        {
```

```
        lblMessage.Text = "I'll roll again.";
        btnRoll_Click(null, null);
    }
    else
    {
        // Stick with roll and pass
        lblMessage.Text = "I pass to you.\r\nYour turn.";
        computerScore = 0;
        computerTotal = v;
        lblComputerTotal.Text =
Convert.ToString(computerTotal);
        lblComputerScore.Text = "";
        whoseTurn = 1;
        btnRoll.Enabled = true;
        btnRoll.Focus();
    }
}
}
```

Run the Project

Save your work. Run the project. You should figure out the game fairly quickly. The computer will decide who goes first. When its your turn, click 'Roll Dice'. After each roll, decide whether to roll again or pass the dice to the computer (click 'Pass Dice'). If you get a frown on any roll, your score will be adjusted accordingly and the dice passed to the computer. When it's the computer's turn, you will watch the computer roll and make its decisions using the same rules. The game is over when either you or the computer has a Total Score of at least 100 points. Click **Stop Game** at any time to stop the game before its end. Here's the middle of a game I played:



Other Things to Try

A first change to Frown would be to make it a two player game - eliminate the computer and play against a friend. You essentially need to have two group boxes and code like that for the human player. You might also like to have an adjustable winning score.

The computer logic used by the program is fairly simple – when it is far behind it tends to take more risks. This logic is in the **timComputer_Tick** event method. Study the logic and see if you can improve upon it. Have your computer play someone else's computer.

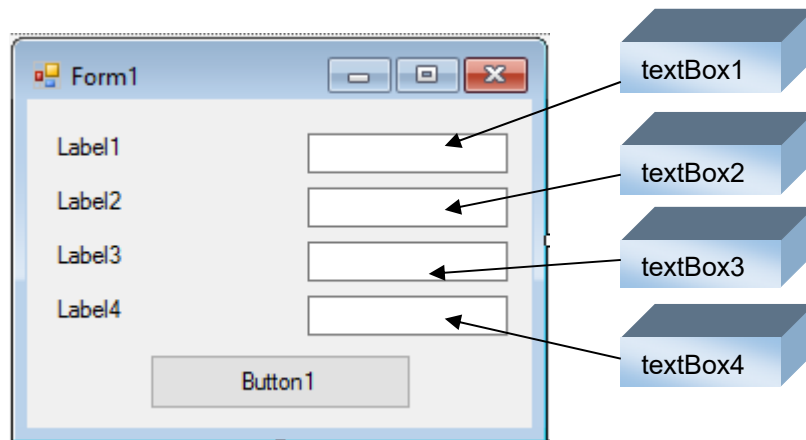
Project 9 - Loan Calculator

Project Design

Do you want to know how much that new car will cost each month or how long it will take to pay off a credit card? This program will do the job. You enter a loan amount, a yearly interest, and a number of months, and the project computes your monthly payment. All entries will be in text boxes and a command button will initiate the payment calculation. The project you are about to build is saved as **Loan** in the project folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Place four label controls and four text boxes on the form. Then place a button on the form. When done, your form should look something like this:



Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Loan Calculator
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

label1 Label:

Property Name	Property Value
Text	Loan Amount

label2 Label:

Property Name	Property Value
Text	Yearly Interest

label3 Label:

Property Name	Property Value
Text	Number of Months

label4 Label:

Property Name	Property Value
Text	Monthly Payment

textBox1 Text Box:

Property Name	Property Value
Name	txtLoan
Text	0
TextAlign	Right

textBox2 Text Box:

Property Name	Property Value
Name	txtInterest
Text	0
TextAlign	Right

textBox3 Text Box:

Property Name	Property Value
Name	txtMonths
Text	0
TextAlign	Right

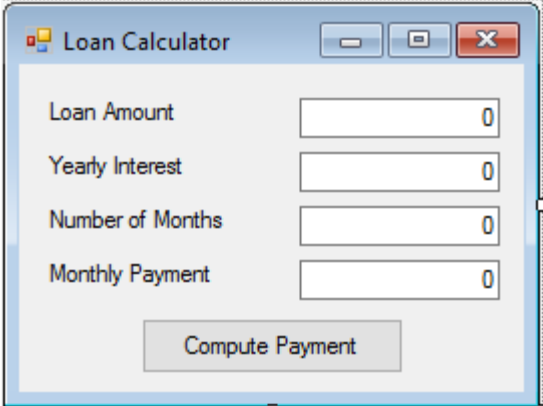
textBox4 Text Box:

Property Name	Property Value
Name	txtPayment
Text	0
TextAlign	Right
BackColor	White
ReadOnly	True
TabStop	False

button1 Button:

Property Name	Property Value
Name	btnCompute
Text	Compute Payment

When done setting properties, my form looks like this:



The image shows a Windows application window titled "Loan Calculator". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. Inside the window, there are four input fields arranged vertically, each with a label to its left and a value of "0" in the text box:

- Loan Amount
- Yearly Interest
- Number of Months
- Monthly Payment

Below these input fields is a single button labeled "Compute Payment".

Write Event Methods

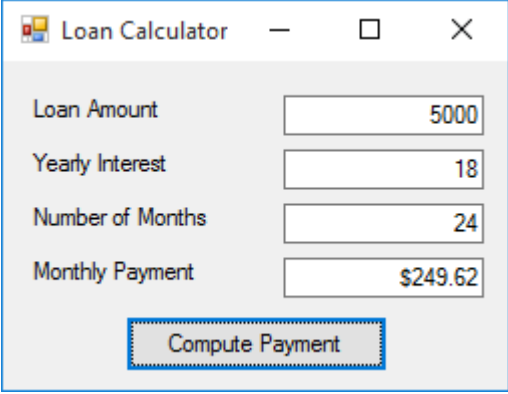
Only one event is needed here - the **Click** event for **btnCompute**. Fill in values in the Loan Amount, Yearly Interest, and Number of Months text boxes, then click **Compute Payment**. The values are read and the payment is computed and displayed.

The **btnCompute_Click** event method:

```
private void btnCompute_Click(object sender, EventArgs e)
{
    double loan, interest, months, payment, multiplier;
    // Read text boxes
    loan = Convert.ToDouble(txtLoan.Text);
    interest = Convert.ToDouble(txtInterest.Text);
    months = Convert.ToDouble(txtMonths.Text);
    // Compute interest multiplier
    multiplier = Math.Pow((1 + interest / 1200), months);
    // Compute payment
    payment = loan * interest * multiplier / (1200 *
(multiplier - 1));
    txtPayment.Text = "$" +
Convert.ToString(String.Format("{0:f2}", payment));
}
```

Run the Project

Save your work. Run the project. Fill in a loan amount, an interest, and a number of months. Click **Compute Payment** to determine and display the monthly payment. Try a loan amount of \$5,000 (don't type in the comma), an interest rate of 18%, and 24 months. Your payment should be \$249.62:



The screenshot shows a Windows application window titled "Loan Calculator". It contains four input fields and one button. The "Loan Amount" field is set to "5000", the "Yearly Interest" field is set to "18", the "Number of Months" field is set to "24", and the "Monthly Payment" field is set to "\$249.62". A button labeled "Compute Payment" is located at the bottom of the form.

Field	Value
Loan Amount	5000
Yearly Interest	18
Number of Months	24
Monthly Payment	\$249.62

Compute Payment

What can you do with this? Well, you can find monthly payments like we just did. Or, try this. Say you have a credit card balance of \$2,000. The interest rate is 15% and you can make \$100 payments each month. Put the 2000 in the loan amount box, the 15 in the interest. Then, try different numbers of months until the computed payment is close to \$100. This will tell you how many months it will take you to pay off the credit card. I got 23 months with payments of \$100.59 each month.

Other Things to Try

If you are going to let others use this program, it needs some improvements. Review the key trapping procedures discussed in Class 10 and make sure users can only type numbers, a decimal point, and a backspace key when using the text boxes for inputs. You need some logic to make sure the user has typed values in all three text boxes (Loan Amount, Yearly Interest, Number of Months). Also, what if the interest rate is zero (a very nice bank!)? The program won't work (try it). You'll need a way to compute payments with zero interest.

Project 10 - Checkbook Balancer

Project Design

This project will help you do that dreaded monthly task of balancing your checkbook. By entering requested information, you can find out just how much money you really have in your account. The project you are about to build is saved as **Checkbook** in the project folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Place three label controls (on one, set **AutoSize** to **False** and make it tall and skinny to use as a dividing line), eight text boxes, and eight buttons on the form. When done, your form should look something like this:

The screenshot shows a Windows Form titled "Form1" with a standard Windows XP-style title bar. The form contains the following controls:

- Buttons:** Eight buttons labeled "Button1" through "Button8". Buttons 1-3 are in a vertical column on the left. Buttons 5-7 are in a vertical column on the right. Buttons 4 and 8 are at the bottom center.
- Text Boxes:** Eight empty text boxes. Four are in a vertical column on the left, four in a vertical column on the right, and two are at the bottom.
- Labels:** Three labels. "Label1" and "Label2" are positioned below the bottom text boxes. "Label3" is a tall, narrow label oriented vertically in the center of the form, acting as a divider.

Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Checkbook Balancer
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

label1 Label:

Property Name	Property Value
AutoSize	False
Text	Adjusted Statement Balance
TextAlign	TopRight

label2 Label:

Property Name	Property Value
AutoSize	False
Text	Adjusted Checkbook Balance

label3 Label:

Property Name	Property Value
AutoSize	False
BackColor	Black
Text	[Blank it out]

textBox1 Text Box:

Property Name	Property Value
Name	txtStmtBalance
Text	0
TextAlign	Right

textBox2 Text Box:

Property Name	Property Value
Name	txtStmtDeposit
Text	0
TextAlign	Right
BackColor	White
ReadOnly	True
TabStop	False

textBox3 Text Box:

Property Name	Property Value
Name	txtStmtCheck
Text	0
TextAlign	Right
BackColor	White
ReadOnly	True
TabStop	False

textBox4 Text Box:

Property Name	Property Value
Name	txtAdjStmtBalance
Text	0
TextAlign	Right
BackColor	White
ReadOnly	True
TabStop	False

textBox5 Text Box:

Property Name	Property Value
Name	txtChkBalance
Text	0
TextAlign	Right

textBox6 Text Box:

Property Name	Property Value
Name	txtChkDeposit
Text	0
TextAlign	Right
BackColor	White
ReadOnly	True
TabStop	False

textBox7 Text Box:

Property Name	Property Value
Name	txtChkCharge
Text	0
TextAlign	Right
BackColor	White
ReadOnly	True
TabStop	False

textBox8 Text Box:

Property Name	Property Value
Name	txtAdjChkBalance
Text	0
TextAlign	Right
BackColor	White
ReadOnly	True
TabStop	False

button1 Button:

Property Name	Property Value
Name	btnStmtBalance
Text	Enter Statement Balance

button2 Button:

Property Name	Property Value
Name	btnStmtDeposit
Text	Add Uncredited Deposit
Enabled	False

button3 Button:

Property Name	Property Value
Name	btnStmtCheck
Text	Subtract Outstanding Check
Enabled	False

button4 Button:

Property Name	Property Value
Name	btnStmtReset
Text	Reset Statement Values

button5 Button:

Property Name	Property Value
Name	btnChkBalance
Text	Enter Checkbook Balance

button6 Button:

Property Name	Property Value
Name	btnChkDeposit
Text	Add Unrecorded Deposit
Enabled	False

button7 Button:

Property Name	Property Value
Name	btnChkCharge
Text	Subtract Service Charge
Enabled	False

button8 Button:

Property Name	Property Value
Name	btnChkReset
Text	Reset Checkbook Values

When done setting properties, my form looks like:

The screenshot shows a Windows application window titled "Checkbook Balancer". The window contains two main columns of input fields and buttons, separated by a thick vertical black line. The left column has four input fields with labels: "Enter Statement Balance", "Add Uncredited Deposit", "Subtract Outstanding Check", and "Adjusted Statement Balance". Each input field contains the number "0". Below these is a button labeled "Reset Statement Values". The right column has four input fields with labels: "Enter Checkbook Balance", "Add Uncredited Deposit", "Subtract Charge / Forgotten Check", and "Adjusted Checkbook Balance". Each input field contains the number "0". Below these is a button labeled "Reset Checkbook Values". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Write Event Methods

Each of the eight command buttons requires a **Click** event. With each click, appropriate adjustments are made to the corresponding account balance.

Add this code to the **general declarations** area:

```
double adjStmtBalance; // adjusted statement balance
double adjChkBalance; // adjusted checkbook balance
```

The **btnStmtBalance_Click** event method:

```
private void btnStmtBalance_Click(object sender, EventArgs e)
{
    // Read entered statement balance
    adjStmtBalance = Convert.ToDouble(txtStmtBalance.Text);
    // Disable balance, enable deposit and check
    btnStmtBalance.Enabled = false;
    btnStmtDeposit.Enabled = true;
    btnStmtCheck.Enabled = true;
    txtStmtBalance.ReadOnly = true;
    txtStmtDeposit.ReadOnly = false;
    txtStmtCheck.ReadOnly = false;
    btnStmtDeposit.Focus();
}
```

The **btnStmtDeposit_Click** event method:

```
private void btnStmtDeposit_Click(object sender, EventArgs e)
{
    // Account for uncredited deposit
    adjStmtBalance = adjStmtBalance +
    Convert.ToDouble(txtStmtDeposit.Text);
    txtAdjStmtBalance.Text = "$" +
    Convert.ToString(String.Format("{0:f2}", adjStmtBalance));
}
```

The **btnStmtCheck_Click** event method:

```
private void btnStmtCheck_Click(object sender, EventArgs e)
{
    // Account for outstanding check
    adjStmtBalance = adjStmtBalance -
Convert.ToDouble(txtStmtCheck.Text);
    txtAdjStmtBalance.Text = "$" +
Convert.ToString(String.Format("{0:f2}", adjStmtBalance));
}
```

The **btnStmtReset_Click** event method:

```
private void btnStmtReset_Click(object sender, EventArgs e)
{
    // Reset statement values to defaults
    adjStmtBalance = 0;
    txtStmtBalance.Text = "0";
    txtStmtDeposit.Text = "0";
    txtStmtCheck.Text = "0";
    txtAdjStmtBalance.Text = "0";
    btnStmtBalance.Enabled = true;
    btnStmtDeposit.Enabled = false;
    btnStmtCheck.Enabled = false;
    txtStmtBalance.ReadOnly = false;
    txtStmtDeposit.ReadOnly = true;
    txtStmtCheck.ReadOnly = true;
    btnStmtBalance.Focus();
}
```

The **btnChkBalance_Click** event method:

```
private void btnChkBalance_Click(object sender, EventArgs e)
{
    // Read entered checkbook balance
    adjChkBalance = Convert.ToDouble(txtChkBalance.Text);
    // Disable balance, enabled deposit and charge
    btnChkBalance.Enabled = false;
    btnChkDeposit.Enabled = true;
    btnChkCharge.Enabled = true;
    txtChkBalance.ReadOnly = true;
    txtChkDeposit.ReadOnly = false;
    txtChkCharge.ReadOnly = false;
    btnChkDeposit.Focus();
}
```

The **btnChkDeposit_Click** event method:

```
private void btnChkDeposit_Click(object sender, EventArgs e)
{
    // Account for unrecorded deposit
    adjChkBalance = adjChkBalance +
    Convert.ToDouble(txtChkDeposit.Text);
    txtAdjChkBalance.Text = "$" +
    Convert.ToString(String.Format("{0:f2}", adjChkBalance));
}
```

The **btnChkCharge_Click** event method:

```
private void btnChkCharge_Click(object sender, EventArgs e)
{
    // Account for service charge
    adjChkBalance = adjChkBalance -
    Convert.ToDouble(txtChkCharge.Text);
    txtAdjChkBalance.Text = "$" +
    Convert.ToString(String.Format("{0:f2}", adjChkBalance));
}
```

The **btnChkReset_Click** event method:

```
private void btnChkReset_Click(object sender, EventArgs e)
{
    // Reset all checkbook values to defaults
    adjChkBalance = 0;
    txtChkBalance.Text = "0";
    txtChkDeposit.Text = "0";
    txtChkCharge.Text = "0";
    txtAdjChkBalance.Text = "0";
    btnChkBalance.Enabled = true;
    btnChkDeposit.Enabled = false;
    btnChkCharge.Enabled = false;
    txtChkBalance.ReadOnly = false;
    txtChkDeposit.ReadOnly = true;
    txtChkCharge.ReadOnly = true;
    btnChkBalance.Focus();
}
```

Run the Project

Save your work. Run the project. Try balancing your latest bank statement with your checkbook - here's the procedure. Start on the left side of the form. Fill in your statement balance and click **Enter Statement Balance**. Next, enter each deposit you have made that is not recorded on the bank statement. After each entry, click **Add Uncredited Deposit**. Next, enter each check you have written that is not listed on the statement. After each check, click **Subtract Outstanding Check**. When done, your **Adjusted Statement Balance** is shown. Clicking **Reset Statement Values** will set everything on the left side back to default values.

Now to the right side of the form. Fill in your checkbook balance and click **Enter Checkbook Balance**. Next, enter each deposit shown on your bank statement that you forgot to enter in your checkbook. After each entry, click **Add Unrecorded Deposit**. Next, enter any service charge the bank may have charged or any check you that you haven't recorded in your checkbook. After each charge, click **Subtract Charge / Forgotten Check**. When done, your **Adjusted Checkbook Balance** is shown. Clicking **Reset Checkbook Values** will set everything on the right side back to default values.

At this point, the adjusted balances at the bottom of the form should be the same. If not, you need to dig deeper into your checks, deposits, and service charges to see what's missing or perhaps accounted for more than once. Here's a run on my account – what do you know? It balanced!!

The screenshot shows a window titled "Checkbook Balancer" with a light gray background. It is divided into two main columns by a thick vertical black line. Each column contains a series of input fields and buttons. On the left side (Statement):

- "Enter Statement Balance" button above a text box containing "318.444".
- "Add Uncredited Deposit" button above a text box containing "100".
- "Subtract Outstanding Check" button above a text box containing "0".
- "Adjusted Statement Balance" label above a text box containing "\$418.44".
- "Reset Statement Values" button at the bottom.

On the right side (Checkbook):

- "Enter Checkbook Balance" button above a text box containing "1193.27".
- "Add Uncredited Deposit" button above a text box containing "100".
- "Subtract Charge / Forgotten Check" button (highlighted with a blue border) above a text box containing "700".
- "Adjusted Checkbook Balance" label above a text box containing "\$418.44".
- "Reset Checkbook Values" button at the bottom.

The adjusted balances on both sides are identical at \$418.44.

Other Things to Try

An interesting and useful modification to this project requires learning about a new control - the **Combo Box**. In this control, you can build up a list of entered information and edit it as you see fit. It would be useful to use combo boxes to store up the uncredited and unrecorded deposits, the outstanding checks, and any service charges. With complete lists, you could edit them as you see fit. This would make the checkbook balancing act an easier task. In time, you can even learn to add printing options to the project.

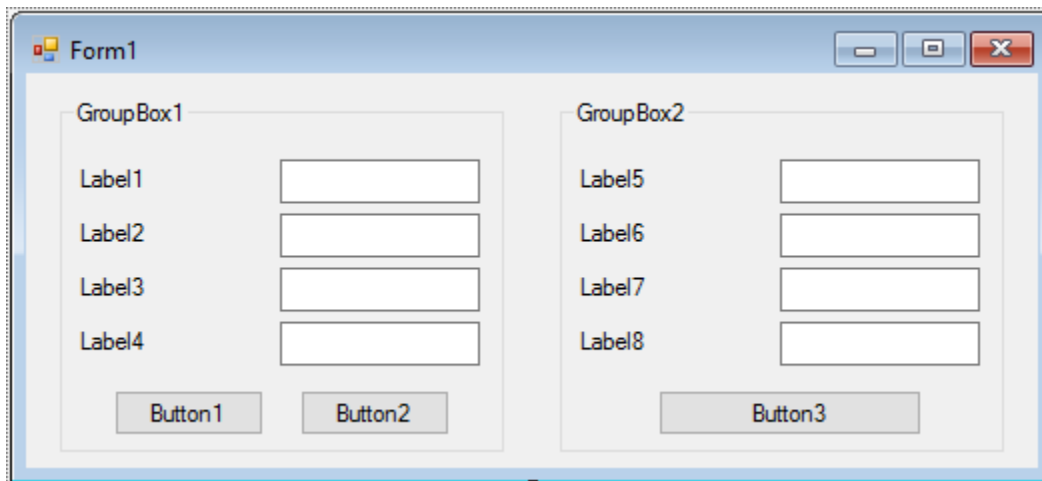
Project 11 – Portfolio Manager

Project Design

In this project, we will build a tool that lets you determine the current value of your stock holdings. You store when you bought a particular stock, how many shares you bought and how much you paid. Then, whenever you want, you enter current values to determine your gain (or possible losses). The project you are about to build is saved as **Portfolio** in the project folder (\BeginVCS\BVCS Projects).

Place Controls on Form

Start a new project in Visual C#. Place two group box controls on the form. In the first group box, place four labels, four text boxes and two buttons. In the second group box, place four labels, four text box controls and a button. When done, your form should look something like this:



The screenshot shows a Windows Form titled "Form1" with a standard Windows XP-style title bar (minimize, maximize, close buttons). The form contains two group boxes, "GroupBox1" and "GroupBox2".

GroupBox1 contains:

- Four labels: "Label1", "Label2", "Label3", and "Label4" arranged vertically.
- Four text boxes arranged vertically, each corresponding to a label.
- Two buttons: "Button1" and "Button2" arranged horizontally at the bottom.

GroupBox2 contains:

- Four labels: "Label5", "Label6", "Label7", and "Label8" arranged vertically.
- Four text boxes arranged vertically, each corresponding to a label.
- One button: "Button3" centered at the bottom.

Set Control Properties

Set the control properties using the properties window:

Form1 Form:

Property Name	Property Value
Text	Portfolio Manager
FormBorderStyle	FixedSingle
StartPosition	CenterScreen

groupBox1 Group Box:

Property Name	Property Value
Name	grpStock
Text	This Stock
Font Size	12
Font Style	Bold

label1 Label:

Property Name	Property Value
Text	Date Purchased
Font Size	8
Font Style	Regular

label2 Label:

Property Name	Property Value
Text	Price/Share
Font Size	8
Font Style	Regular

label3 Label:

Property Name	Property Value
Text	Number of Shares
Font Size	8
Font Style	Regular

label4 Label:

Property Name	Property Value
Text	Price Paid
Font Size	8
Font Style	Regular

textBox1 Text Box:

Property Name	Property Value
Name	txtDate
TextAlign	Right
BackColor	White
Font Size	10
Font Style	Regular
ReadOnly	True
TabStop	False

textBox2 Text Box:

Property Name	Property Value
Name	txtPrice
TextAlign	Right
BackColor	White
Font Size	10
Font Style	Regular
ReadOnly	True
TabStop	False

textBox3 Text Box:

Property Name	Property Value
Name	txtShares
TextAlign	Right
BackColor	White
Font Size	10
Font Style	Regular
ReadOnly	True
TabStop	False

textBox4 Text Box:

Property Name	Property Value
Name	txtPaid
TextAlign	Right
BackColor	White
Font Size	10
Font Style	Regular
ReadOnly	True
TabStop	False

button1 Button:

Property Name	Property Value
Name	btnPrevious
Text	Previous
Font Size	8
Font Style	Regular

button2 Button:

Property Name	Property Value
Name	btnNext
Text	Next
Font Size	8
Font Style	Regular

groupBox2 Group Box:

Property Name	Property Value
Name	grpValue
Text	Current Value
Font Size	12
FontBold	True

label5 Label:

Property Name	Property Value
Text	Today's Date
Font Size	8
Font Style	Regular

label6 Label:

Property Name	Property Value
Text	Today's Price
Font Size	8
Font Style	Regular

label7 Label:

Property Name	Property Value
Text	Yearly Return
Font Size	8
Font Style	Regular

label8 Label:

Property Name	Property Value
Text	Today's Value
Font Size	8
Font Style	Regular

textBox1 Text Box:

Property Name	Property Value
Name	txtTodayDate
TextAlign	Right
BackColor	White
Font Size	10
Font Style	Regular
ReadOnly	True
TabStop	False

textBox2 Text Box:

Property Name	Property Value
Name	txtToday
TextAlign	Right
Font Size	10
Font Style	Regular

textBox3 Text Box:

Property Name	Property Value
Name	txtReturn
TextAlign	Right
BackColor	White
Font Size	10
Font Style	Regular
ReadOnly	True
TabStop	False

textBox4 Text Box:

Property Name	Property Value
Name	txtValue
TextAlign	Right
BackColor	White
Font Size	10
Font Style	Regular
ReadOnly	True
TabStop	False

button3 Button:

Property Name	Property Value
Name	btnReturn
Text	Compute Return

When done setting properties, my form looks like this:

The screenshot shows a Windows application window titled "Portfolio". The window contains a form with two main sections: "This Stock" and "Current Value".

This Stock

- Date Purchased:
- Price/Share:
- Number of Shares:
- Price Paid:

Below these fields are two buttons: "Previous" and "Next".

Current Value

- Today's Date:
- Today's Price:
- Yearly Return:
- Today's Value:

Below these fields is a button: "Compute Return".

Write Event Methods

In this program, you need to store information about your stocks (date purchased, purchase price and shares owned) in data arrays (the form **Load** method). Then, you use the **Previous** and **Next** buttons to view each stock. For the displayed stock, if you type in the current price (**Today's Price**) and click **Compute Return**, you will be shown the current value and yearly return for that stock.

In this project, we introduce an idea that's used all the time in computer programming. Whenever, there is a certain segment of code that needs to be repeated and used in various parts of a project, we put the corresponding code in something called a **general method**. This saves us from having to repeat code in different locations – a maintenance headache. A general method is identical in use to an event method, with the only difference being it is not invoked by some control event. We control invocation of a general method by **calling** it. In this project, we will use a general method to display the stock information after pressing the **Previous** or **Next** button. Look for the code (method is named **ShowStock**) and see how easy it is to use.

Add this code to the **general declarations** area:

```
int numberStocks;  
int currentStock;  
DateTime[] stockDate = new DateTime[25];  
string[] stockName = new string[25];  
double[] stockPrice = new double[25];  
int[] stockShares = new int[25];
```

The **Form1_Load** event method:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Load stock Information
    numberStocks = 6;
    stockDate[0] = new DateTime(2002, 2, 10); stockName[0] =
    "Only Go Up";
    stockPrice[0] = 10; stockShares[0] = 50;
    stockDate[1] = new DateTime(2001, 2, 1); stockName[1] =
    "Big Deal";
    stockPrice[1] = 10; stockShares[1] = 100;
    stockDate[2] = new DateTime(2001, 3, 1); stockName[2] =
    "Web Winner";
    stockPrice[2] = 20; stockShares[2] = 300;
    stockDate[3] = new DateTime(2003, 4, 10); stockName[3] =
    "Little Blue";
    stockPrice[3] = 15; stockShares[3] = 200;
    stockDate[4] = new DateTime(1999, 5, 21); stockName[4] =
    "My Company";
    stockPrice[4] = 40; stockShares[4] = 400;
    stockDate[5] = new DateTime(2000, 11, 1); stockName[5] =
    "Your Company";
    stockPrice[5] = 30; stockShares[5] = 200;
    txtTodayDate.Text = DateTime.Today.ToShortDateString();
    currentStock = 0;
    this.Show();
    ShowStock();
}
```

The **btnPrevious_Click** event method:

```
private void btnPrevious_Click(object sender, EventArgs e)
{
    // display previous stock
    if (currentStock != 0)
    {
        currentStock--;
        ShowStock();
    }
}
```

The **btnNext_Click** event method:

```
private void btnNext_Click(object sender, EventArgs e)
{
    // display next stock
    if (currentStock != numberStocks - 1)
    {
        currentStock++;
        ShowStock();
    }
}
```

Next, we give the code for the **general method** called **ShowStock**. To type this in the code window, go to any line after an existing method. Type the framework for the method:

```
private void ShowStock()
{

```



Type code here

```
}
```

Type the code between the two curly braces following the header you typed.

The complete method is:

```
private void ShowStock()
{
    // Change displayed stock
    grpstock.Text = stockName[currentStock];
    txtDate.Text =
stockDate[currentStock].ToShortDateString();
    txtPrice.Text =
Convert.ToString(stockPrice[currentStock]);
    txtShares.Text =
Convert.ToString(stockShares[currentStock]);
    txtPaid.Text = String.Format("{0:f2}",
stockPrice[currentStock] * stockShares[currentStock]);
    // Allow computation of return
    txtToday.Text = "";
    txtValue.Text = "0.00";
    txtReturn.Text = "0.00%";
    txtToday.Focus();
}
```

The `txtToday_KeyPress` event method:

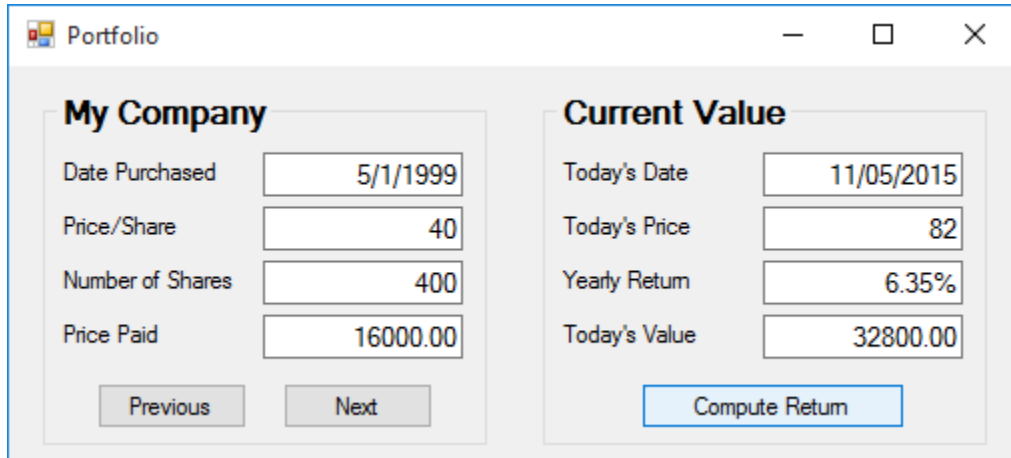
```
private void txtToday_KeyPress(object sender,
KeyPressEventArgs e)
{
    // Only allow numbers, decimal backspace
    if ((int) e.KeyChar == 13)
    {
        // Return key pressed
        btnReturn.PerformClick();
    }
    else if ((e.KeyChar >= '0' && e.KeyChar <= '9') ||
e.KeyChar == '.' || (int) e.KeyChar == 8)
    {
        e.Handled = false;
    }
    else
    {
        e.Handled = true;
    }
}
```

The **btnReturn_Click** event method:

```
private void btnReturn_Click(object sender, EventArgs e)
{
    // compute todays value and percent return
    double p, v, r;
    p = Convert.ToDouble(txtToday.Text);
    v = p * stockShares[currentStock];
    txtValue.Text = String.Format("{0:f2}", v);
    // Daily increase
    TimeSpan diff = DateTime.Today - stockDate[currentStock];
    r = (v / Convert.ToDouble(txtPaid.Text) - 1) / diff.Days;
    // Yearly return
    r = 100 * (365 * r);
    txtReturn.Text = String.Format("{0:f2}", r) + "%";
}
```

Run the Project

Save your work. Run the project. Click the **Previous** and **Next** buttons to view the five stocks stored in the program (you can edit this information in the **Form1_Load** method to reflect your holdings). Make sure you understand the use of the general method (**ShowStock**). For a particular stock, type the current selling price in the displayed text box and click **Compute Return**. The yearly return percentage and current value of that particular stock to your portfolio is displayed. Here's a run I made:



The screenshot shows a Windows application window titled "Portfolio". It contains two main sections: "My Company" and "Current Value".

My Company		Current Value	
Date Purchased	5/1/1999	Today's Date	11/05/2015
Price/Share	40	Today's Price	82
Number of Shares	400	Yearly Return	6.35%
Price Paid	16000.00	Today's Value	32800.00
<input type="button" value="Previous"/> <input type="button" value="Next"/>		<input type="button" value="Compute Return"/>	

Other Things to Try

It's a hassle to have to store your holdings in the various arrays. You have to change the code every time you buy new stock or sell old stock. It would be nice to be able to save your holding information on a disk file. Then, it could be read in each time you run the program and any changes saved back to disk. With such saving capabilities, you could also modify the program to allow changing the number of shares you hold of a particular stock, allow addition of new stocks and deletion of old stocks. Accessing files on disk is an advanced topic you might like to study.

As written, the program gives returns on individual stocks. Try to write a summary function that computes the overall return on all the stocks in your current portfolio. I'm sure you can think of other changes to this program. Try them out.

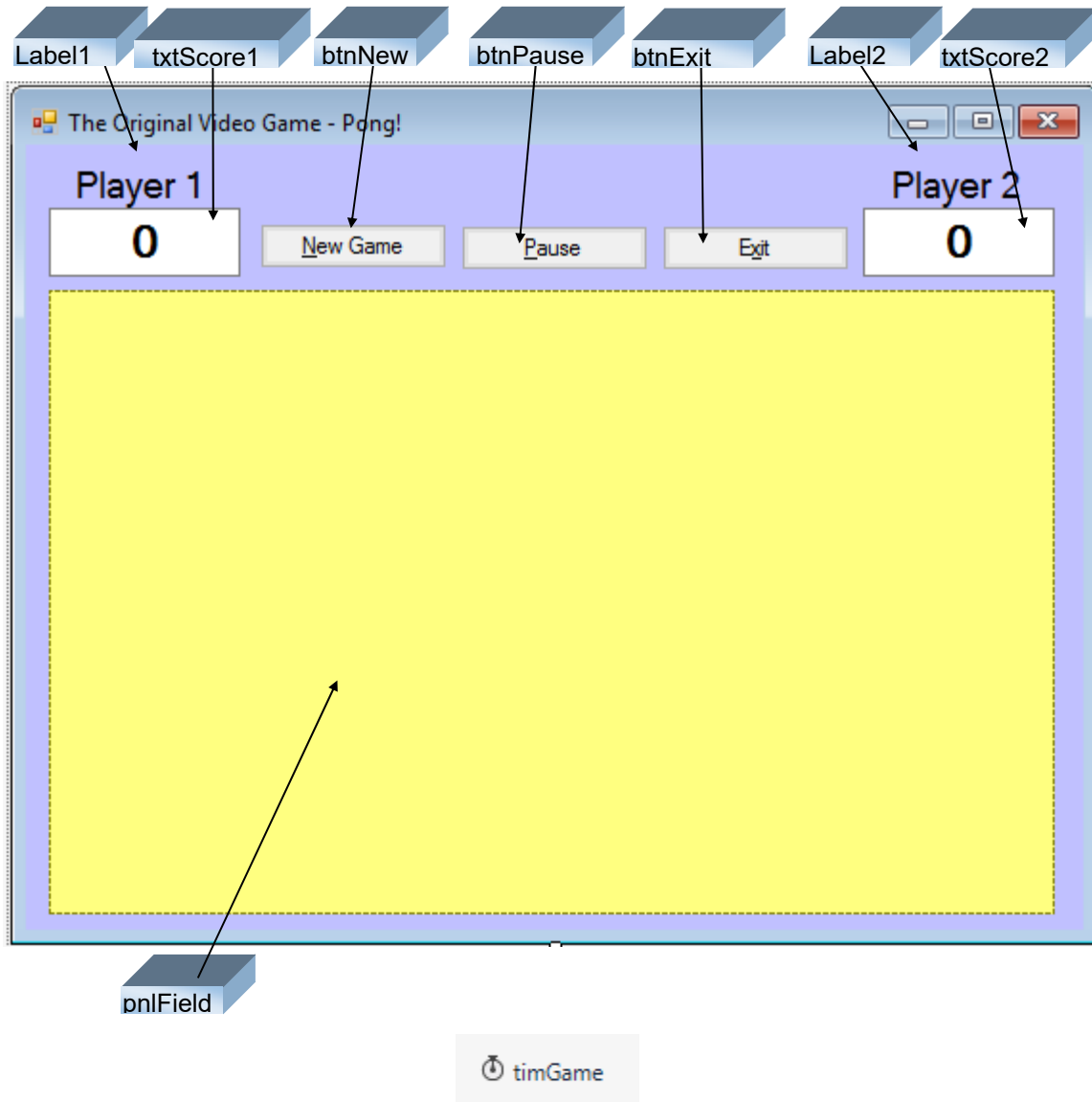
Bonus Project 12 - Pong!

In the early 1970's, while Bill Gates and Paul Allen were still in high school, a man named Nolan Bushnell began the video game revolution. He invented a very simple game - a computer version of Ping Pong. There were two paddles, one on each side of the screen. Players then bounced the ball back and forth. If you missed the ball, the other player got a point.

This first game was called Pong. And, Nolan Bushnell was the founder of Atari - the biggest video game maker for many years. (Nolan Bushnell also founded Chucky Cheese's Pizza Parlors, but that's another story!) In this bonus project, I give you my version of Pong written with Visual C#. I don't expect you to build this project, but you can if you want. Just load the project (named **Pong**) and run it. Skim through the code - you should be able to understand a lot of it. The idea of giving you this project is to let you see what can be done with Visual C#.

In this version of Pong, a ball moves from one end of a panel to the other, bouncing off side walls. Players try to deflect the ball at each end using a controllable paddle. In my simple game, the left paddle is controlled with the A and Z keys on the keyboard, while the right paddle is controlled with the K and M keys (detected using KeyPress events). My solution freely borrows code and techniques from several reference sources. The project relies heavily on lots of coding techniques you haven't seen. You will learn about these as you progress in your Visual C# studies.

Start Visual C#. Open the project named **Pong** in the project folder (**\BeginVCS\BVCS Projects**). Look at the form. Here's what my finished form looks like (with control names identified):



The graphics (paddles and ball) are loaded from files stored with the application. Try to identify controls you have seen before. Go to the properties window and look at the assigned properties. Run the project and play the game with someone. In particular, notice the cool sounds (if you have a sound card in your computer). This is something that should be a part of any Visual C# project – these sounds are also loaded from files. Have fun with Pong! Can you believe people used to spend hours mesmerized by this game? It seems very tame compared to today's video games, but it holds a warm spot in many people's gaming hearts. Here's a game I was playing:

