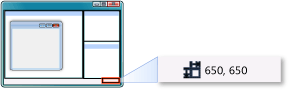
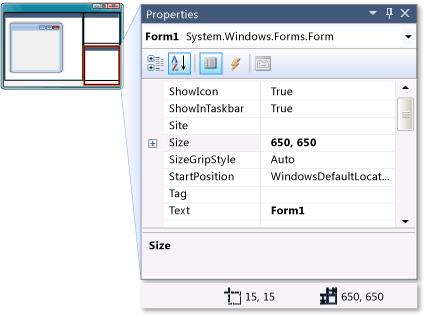
To create a project and add a Panel container

1. On the **File** menu, click **New Project**.
2. If you’re not using Visual Studio Express, you need to select a language first. From the **Installed Templates**list, select either **C#** or **Visual Basic**
3. Click the **Windows Forms Application** icon, and then type **Maze** as the name.
4. Set the form properties:
   1. Resize your form by using your pointer to drag the lower-right corner. Watch the lower-right corner of the integrated development environment (IDE). The size of the form appears in the status bar. Keep dragging until your form is 650 pixels wide and tall. You can build a smaller or bigger maze, so make the form any size you want.

Size in status bar



* 1. After your form is the right size, set the **Text** property to **Maze**.
  2. So that the user cannot resize the form, set the **FormBorderStyle** property to **Fixed3D**.
  3. Disable the **Maximize** button in the form's title bar by setting the **MaximizeBox** property to **False**.

Now you have a form that's a fixed size, which the user can't maximize.

|  |
| --- |
| **NoteNote** |
| When you create a new form, it's set up by default to let the user resize it in two ways: The user can either drag the sides or corners of the form or click the **Maximize** button to maximize the form. If you want to be sure a user can't resize your form, disable both of these options. Setting the **FormBorderStyle** property to any of the fixed styles prevents the user from resizing it, but the user can still click the **Maximize** button. That's why you also need to disable the **MaximizeBox** property. |

Next, you want to create a playing field, where you build the maze. You use a **Panel** control for that. A panel is a type of container control that lets you lay out a group of controls. Unlike some of the other containers (like the **TableLayoutPanel** container and the **FlowLayoutPanel** container), a panel doesn't rearrange the controls that it contains. That gives you the freedom to position the controls wherever you want, but unlike the TableLayoutPanel or FlowLayoutPanel, a panel isn't helpful when the user resizes the window.

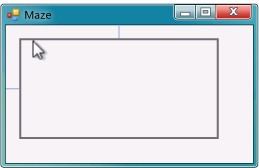
1. Go to the **Containers** group in the Toolbox and double-click **Panel** to add a panel to your form. When your panel is selected, you should see a move handle icon in its upper-left corner, which appears as follows.

Move handle

Move handle

1. Drag the panel until it's a small distance away from the upper-left corner of the form. As you drag it, you should notice a useful feature of the IDE: As soon as the panel is a certain distance from the top or the left edge of the form, it snaps into place, and a blue spacer line appears between the edge of the panel and the edge of the form. You can use this to easily align your panel so that its edges are all exactly the same distance from the edge of the form. As soon as you see the top and left blue spacer lines, release the mouse button to drop the panel in place. The blue spacer lines appear as follows.

Blue spacer lines



Drag the lower-right drag handle until the panel snaps into place on the right and bottom.

1. Because you want the user to see the edge of the maze, you need to give it a visible border. Select the panel and set its **BorderStyle** property to **Fixed3D**.
2. Save the project by clicking the **Save All** toolbar button, which appears as follows.

Save All button

Save All toolbar button

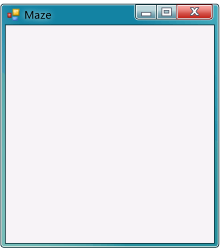
1. To run your program, press F5 or click the **Start Debugging** toolbar button, which appears as follows.

Start Debugging toolbar button

Start Debugging toolbar button

When running, your form should look like the following picture.

Initial maze form

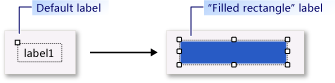


1. Before going to the next tutorial step, stop your program by either closing the form or clicking the **Stop Debugging** toolbar button on the **Debug** toolbar. (The IDE stays in read-only mode while your program runs.)

To build your maze using labels

1. In Windows Forms Designer, go to the **Common Controls** group in the Toolbox and double-click **Label** to make the IDE add a label to your form.
2. Set a few properties so that the label becomes a rectangle, which you can resize:
   * Set the **AutoSize** property to **False**.
   * Set the **BackColor** property to any color you like. (For this tutorial, **RoyalBlue** is selected from the **Web color** tab.)
   * Change the **Text** property so that it's empty by selecting the text **label1** and deleting it.

Label as a filled rectangle

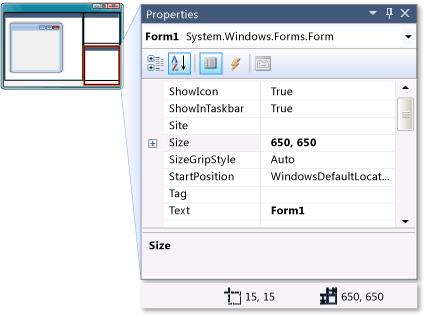


Your **Label** control should now be a filled rectangle.

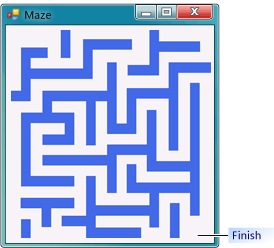
|  |
| --- |
| **NoteNote** |
| This may seem somewhat unusual because a **Label** control is meant to be used as a label. In this case, the label is used as a drawing block because it works. An important part of programming is recognizing when a tool in your toolbox (or, in this case, your IDE's Toolbox) works for the job, even if it's not the job it was originally intended for. |

1. Now you can be creative when building your maze. Copy your label by selecting it, and from the **Edit** menu, select **Copy** (or press Ctrl+C). Then, paste it several times. From the **Edit** menu, select **Paste** (or press Ctrl+V). This should provide horizontal maze walls. Take one of the walls and drag it so that it's tall and narrow. Copy and paste it a few times to provide vertical walls.
2. Drag the labels around your panel and create your maze. Don't make the passages too narrow, or your game will be too difficult to play. Leave extra space in the upper-left corner, because that's where the player starts the maze.

|  |
| --- |
| **NoteNote** |
| As you may remember, the size of the form appears in the IDE's status bar when you resize it. The IDE does the same thing when you resize your labels or any other control. You can use this to be sure all of the maze walls are the same width, if you want.  The IDE's alignment bars that you used to position the panel are also useful when you position the maze walls. You can also use the arrow keys on the keyboard to make fine adjustments to the position of the control that's currently selected. The following picture shows the size in the status bar. |

1. Size in status bar
2. 
3. After you lay out your maze, go to the **Common Controls** group in the Toolbox and double-click **Label** once again. Use the **(Name)** line in the **Properties** window to name it **finishLabel**, and change its **Text** property to**Finish**.
4. Drag your new **Finish** label to the end of the maze. That's the target that the user needs to hit.
5. Save your project and run your program again. The following is an example of a finished maze form. (Your maze will look different.)

Finished maze form



To end the game

1. Select the **finishLabel** control, and then click the **Event** icon at the top of the **Properties** window, which is shaped like a lightning bolt. When you click it, instead of showing the control's properties, it shows the control's events. You can return to the list of properties by clicking the **Property** icon. For now, keep the**Properties** window as is, so it's showing all of the events for the **finishLabel** control. Scroll down to the MouseEnter event. The icons and the MouseEnter event appear as follows.

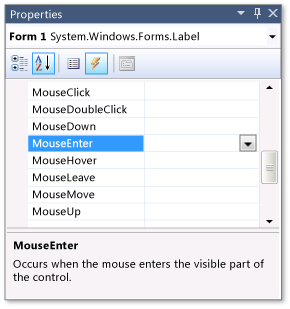
Event icon

Event icon

Property icon

Property icon

MouseEnter event



1. Double-click the word **MouseEnter**. After you do, the IDE automatically adds an event handler method to your form and shows it to you in the code editor, as follows.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492161(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

private void finishLabel\_MouseEnter(object sender, EventArgs e)

{

}

This event handler method runs every time the mouse pointer enters the label.

1. You want the program to open a message box that shows "Congratulations," and then you want the program to close. To do that, add lines of code (with a comment), as follows.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492161(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

private void finishLabel\_MouseEnter(object sender, EventArgs e)

{

// Show a congratulatory MessageBox, then close the form.

MessageBox.Show("Congratulations!");

Close();

}

|  |
| --- |
| **NoteNote** |
| Your **finishLabel\_MouseEnter()** method has two statements. The first statement is calling a method called**Show()**, which opens a message box that contains whatever text you put inside the parentheses. |

1. You can learn more about what's happening by using the IDE to explore your code. Take your mouse pointer and position it so it's over the word **MessageBox**. You should see the following tooltip.

Tooltip

Tooltip

|  |
| --- |
| **NoteNote** |
| The IDE shows that there's a class called **System.Windows.Forms.MessageBox**, and the **Show()** method that you called is inside that class. You don't need a complete understanding to make the message box work, but additional information can be helpful.  Regarding the second statement, every form has a built-in method called **Close()** that causes the form to close. Some programs have several windows that the user can switch between. When working on a program like that, it closes the current window, but leaves the rest of the program running. (For example, if you have several Microsoft Office Word documents open at the same time, closing one document window closes that document, but Office Word stays open.) However, in a program like this one, where there's only one window, closing that window causes the program to stop running, so closing the form closes your program. |

1. Save and run your program. Move your mouse pointer over the **Finish** label. It should open the message, and then close the program.

To add a method to restart the game

1. Go to the code for the form by right-clicking **Form1.cs** in **Solution Explorer** and selecting **View Code** from the menu.
2. You should see the **finishLabel\_MouseEnter()** method that you added. Just below that method, add a new**MoveToStart()** method.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492145(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

private void MoveToStart()

{

Point startingPoint = panel1.Location;

startingPoint.Offset(10, 10);

Cursor.Position = PointToScreen(startingPoint);

}

1. There's a special type of comment that you can add above any method, and the IDE can help you add it. Put your cursor on the line above the new method. In Visual C#, add three slash marks (///). In Visual Basic, add three single quotation marks ('''). The IDE automatically fills in the following text.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492145(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

/// <summary>

///

/// </summary>

private void MoveToStart()

{

Point startingPoint = panel1.Location;

startingPoint.Offset(10, 10);

Cursor.Position = PointToScreen(startingPoint);

}

1. On the line between the two summary tags, fill in the following comment. (After you press ENTER, the IDE automatically adds a new line with either three slash marks (///) or three single quotation marks ('''), depending on your programming language, so you can continue your comment.)

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492145(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

/// <summary>

/// Move the pointer to a point 10 pixels down and to the right

/// of the starting point in the upper-left corner of the maze.

/// </summary>

|  |
| --- |
| **NoteNote** |
| You just added an XML comment. As you may remember, the IDE showed you information in a tooltip when you paused above the word **MessageBox**. The IDE fills in tooltips for your methods automatically. Anything you put into an XML comment appears in the IDE's tooltip, as well as in the **IntelliSense** window. With a program with many methods, that can be useful. Also, if you put a wall that's 10 pixels down and to the right of the upper-left corner of the panel, you can change (10, 10) in the code. Experiment with different numbers until you find a pointer starting point that works for your maze. |

1. After you add your method, you need to call it. Because you want your program to move the pointer over the starting point as soon as the program starts, you should call the method as soon as the form starts. For Visual C#, look for the following method in your form's code.

C#

public Form1()

{

InitializeComponent();

}

For Visual Basic, add this method in your form's code. Before the **finishLabel\_MouseEnter** method, start typing the following code.

VB

Public Sub New()

When you press the ENTER key to move to the next line, IntelliSense should make the following code appear to complete the method.

VB

Public Sub New()

' This call is required by Windows Forms Designer.

InitializeComponent()

' Add any initialization after the InitializeComponent() call.

End Sub

This is a special method called the constructor. It gets executed once, when your form is created. Right now, all it does is call a method called **InitializeComponent()**. You will add a line to it to call the new**MoveToStart()** method that you just wrote. Before you continue, consider what to add to your program to get it to call the **MoveToStart()** method immediately after it calls the **InitializeComponent()** method.

|  |
| --- |
| **NoteNote** |
| The **InitializeComponent()** method in your form's constructor is a method that the IDE wrote. It adds all of the controls and components to the form, and sets up their properties. Any time you change any of the properties of your form or its controls, the IDE alters that method. You can look at it by opening the file Form1.Designer.cs from **Solution Explorer**. You don't need to edit the contents of the**InitializeComponent()** method. The IDE takes care of this based on the form that you created in the Design view. |

1. Add a call to the **MoveToStart()** method immediately after it calls the **InitializeComponent()** method. Your form code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492145(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

namespace Maze

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

MoveToStart();

}

private void finishLabel\_MouseEnter(object sender, EventArgs e)

{

// Show a congratulatory MessageBox, then close the form.

MessageBox.Show("Congratulations!");

Close();

}

/// <summary>

/// Move the pointer to a point 10 pixels down and to the right

/// of the starting point in the upper-left corner of the maze.

/// </summary>

private void MoveToStart()

{

Point startingPoint = panel1.Location;

startingPoint.Offset(10, 10);

Cursor.Position = PointToScreen(startingPoint);

}

}

}

Note the call to the **MoveToStart()** method underneath **InitializeComponent()**. If you're programming in Visual C#, remember to end that line with a semicolon (;), or your program won't build.

1. Now save your program and run it. As soon as the program starts, your pointer should automatically be repositioned slightly down and to the right of the upper-left corner of the panel.

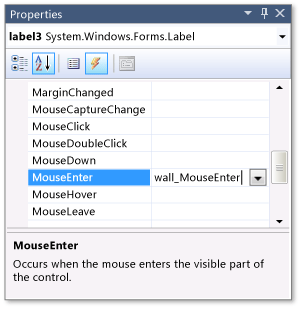
To add a MouseEnter event handler for each wall

1. Go to Windows Forms Designer and click any of your newly added walls.
2. Go to the **Properties** window and click the **Event** icon to display the events for that wall. Scroll down to the MouseEnter event. Instead of double-clicking it, type the text **wall\_MouseEnter**, and then press ENTER. The**Event** icon and **Properties** window appear as follows.

Event icon

Event icon

Properties window showing MouseEnter event



|  |
| --- |
| **NoteNote** |
| When you type the event name directly into the event table in the **Properties** window, you direct the IDE to create an event handler with that name and connect it to the control's event. Often, you want the IDE to choose event names, because the names are logical, and using names makes it easier for others to read and understand your code. When the IDE chooses a name for an event handler, it uses the name of the control and the name of the event. In this case, you didn't change the default names of your walls, which are **label4**, **label18**, **label25**, and so on. So if you click a wall named **label12**, the IDE would have named the event handler **label12\_MouseEnter**. By typing the name **wall\_MouseEnter**, you're choosing a more applicable name. This is especially important when you use one event handler for multiple controls, which is what you do later in this tutorial. |

1. After you press ENTER, the IDE adds a new event handler for you and connects it to that wall's MouseEnter event. The newly added code should appear in your code editor as follows. In Visual Basic, the specific label may not be Label8, as shown in the code.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492149(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

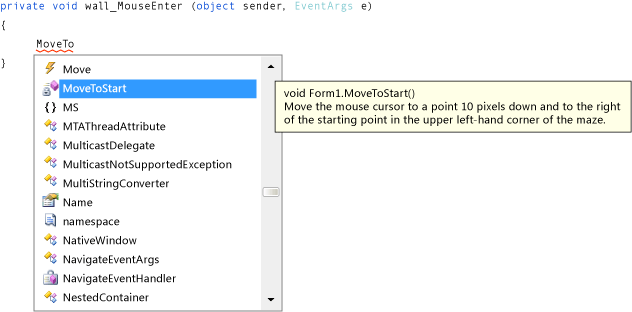
private void wall\_MouseEnter(object sender, EventArgs e)

{

}

1. Next, add a call to your **MoveToStart()** method, along with a comment explaining the method. Start by going to your method and adding the statement **MoveToStart()**. An **IntelliSense** window opens, and the following appears.

IntelliSense window



When you added your **MoveToStart()** method, the IDE added it to the **IntelliSense** window. The XML comment that you added appears in the tooltip. This is useful when you write programs with numerous methods.

1. Press TAB to direct IntelliSense to complete the method name. If you're writing Visual C# code, remember to add the semicolon (;) at the end of the statement. Then add a comment above the statement. Your code should look like the following. In Visual Basic, the specific label may not be Label8, as shown in the code.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492149(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

private void wall\_MouseEnter(object sender, EventArgs e)

{

// When the mouse pointer hits a wall or enters the panel,

// call the MoveToStart() method.

MoveToStart();

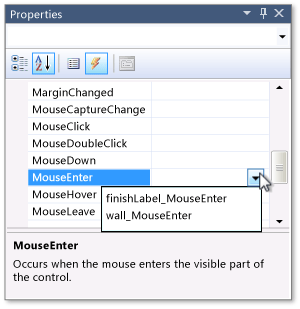
}

1. Save and run your program. Move your mouse pointer over the wall that you connected the event handler to. (If you don't remember which one you chose, move your mouse pointer over each wall until you find the right one.) As soon as you touch it, it should send your mouse pointer back to the start.

Next, you want to do the same for the rest of the walls. You could write the same MouseEnter event handler for each of the walls. But that process would be lengthy, would result in multiple lines of the same code in your program, and would be difficult to change. The IDE provides an easier way to connect the same event handler to all of the walls.

1. Go to Windows Forms Designer, and from the **Edit** menu, click **Select All**.
2. Hold down the CTRL key, and then click the **Finish** label to clear the selection. This should leave all of the walls and the panel selected.
3. Now go to the event table on the **Properties** window. Scroll down to the MouseEnter event and click the edit box next to it. You should see a drop-down arrow. If you click the arrow, you see a list of all of the event handlers in your program that you can choose for this event. In this case, you should see the finishLabel\_MouseEnter event handler that you added earlier, and the wall\_MouseEnter one that you just wrote, as shown in the following picture.

MouseEnter event with event handlers



1. Select **wall\_MouseEnter**. (If you select the wrong event handler or accidentally add a new one, you can select all of the walls and the panel again, and then choose the right method.)
2. Now your maze game should be more fun. Try saving it and running it. If your pointer hits a wall or if you move your pointer out of the maze and back in again, the program should automatically reposition the pointer at the starting point of the maze.

To add a SoundPlayer for noise

1. Start by adding a SoundPlayer to your form's code, just above the constructor.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492153(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

public partial class Form1 : Form

{

// This SoundPlayer plays a sound whenever the player hits a wall.

System.Media.SoundPlayer startSoundPlayer = new System.Media.SoundPlayer(@"C:\Windows\Media\chord.wav");

public Form1()

{

InitializeComponent();

MoveToStart();

}

|  |
| --- |
| **NoteNote** |
| The first line (public partial class Form1 : Form) has appeared several times before. It's important, because it includes the class keyword. The class keyword appears many times, because a class is a basic building block of any program. |

1. Earlier, you put your mouse pointer over the word MessageBox in the statement**MessageBox.Show("Congratulations!");**, to make the IDE open a tooltip. Do this again now, but take a closer look at the first line, which appears as follows.

Tooltip

Tooltip

|  |
| --- |
| **NoteNote** |
| The class keyword appears in the first line. It appears frequently because your code is organized into classes as follows: Your program has classes, each class has methods, and each method has statements. There are numerous built-in classes, such as **MessageBox**. The **MessageBox** class has a method called**Show()**, and when called, it executes statements that open a message box. You have also worked with**Button**, **Label**, and **Panel** classes. When you set their properties, you worked with another aspect of classes: A class can have properties as well as methods, and setting those properties can cause the class to execute statements that change behavior. |

As you may realize, **SoundPlayer** is a class that plays a sound. When you create a SoundPlayer with the **new**keyword, it loads a sound from a file, which you can play using its **Play()** method. You will use this SoundPlayer to play the Windows Chord sound when the player starts a new game, or when the pointer touches a wall and the player has to start over. (That's why it's called startSoundPlayer.)

1. If you want to use different sounds, replace the path between the quotation marks in the new statement (C:\Windows\Media\chord.wav) with the path of the sound file that you want to use.

When you build your form in Windows Forms Designer, you use the IDE to help you create your own class, in this case, a class called **Form1**. When you added that line of code above your constructor, you added a new SoundPlayer to your form, just like you previously added a button or a label. The statement is located outside of the methods so that the SoundPlayer can be accessed by more than one method. That's why you had to put the new statement inside your form's code but outside of its methods. You named it startSoundPlayer, the same way you named one of your **Label** controls finishLabel.

After you add the statement to create a new SoundPlayer and call it startSoundPlayer, it appears in the**IntelliSense** window, just like labels, buttons, and other controls.

This may seem complicated, but it's similar to what you did previously in the IDE. For example, when you use the IDE's Toolbox to add a button or label to the form, the IDE adds lines of code automatically that are used to create a new button or a new label. You do the same now, except this time, you create a SoundPlayer. (A second SoundPlayer is created in the next tutorial step.)

To play sounds

1. Start by adding a second SoundPlayer to play the Windows Tada sound. Your game will play this sound when the player reaches the **Finish** label.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492141(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

public partial class Form1 : Form

{

// This SoundPlayer plays a sound whenever the player hits a wall.

System.Media.SoundPlayer startSoundPlayer = new System.Media.SoundPlayer(@"C:\Windows\Media\chord.wav");

// This SoundPlayer plays a sound when the player finishes the game.

System.Media.SoundPlayer finishSoundPlayer = new System.Media.SoundPlayer(@"C:\Windows\Media\tada.wav");

public Form1()

{

InitializeComponent();

MoveToStart();

}

1. Both SoundPlayers are now added to your form. Add a **Play()** method to call the SoundPlayer to play the sound at the appropriate time. You want a sound to play when the user hits a wall. So add the statement**startSoundPlayer.Play();** to your **MoveToStart()** method. Remember to update the comment. The final method looks like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492141(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

/// <summary>

/// Play a sound, then move the mouse pointer to a point 10 pixels down and to

/// the right of the starting point in the upper-left corner of the maze.

/// </summary>

private void MoveToStart()

{

startSoundPlayer.Play();

Point startingPoint = panel1.Location;

startingPoint.Offset(10, 10);

Cursor.Position = PointToScreen(startingPoint);

}

1. Add the statement **finishSoundPlayer.Play();** to the **Finish** label MouseEnter event handler. Remember to update the comment, because you're changing the code, as follows.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492141(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

private void finishLabel\_MouseEnter(object sender, EventArgs e)

{

// Play a sound, show a congratulatory MessageBox, then close the form.

finishSoundPlayer.Play();

MessageBox.Show("Congratulations!");

Close();

}

To run your program

1. Save your program, and then start it.
2. Be sure your mouse pointer is positioned at the beginning of the maze.
3. Move your mouse pointer through the maze. Touch a wall, and be sure a sound plays and the mouse pointer is sent back to the start.
4. Move your mouse pointer outside the maze. Then, move the mouse pointer back into the panel, and verify that the mouse pointer is sent back to the start.

|  |
| --- |
| **NoteNote** |
| When testing, you need to be sure that everything in the program works. You want to test to be sure that the **Finish** label MouseEnter event handler plays the Tada sound, opens a congratulatory message box, and closes the game. To avoid going through the entire maze, you can temporarily turn off the panel's MouseEnter event handler. That way, you can move your mouse pointer outside of the maze and position it on the **Finish** label without being sent back to the starting point. |

1. Select the panel, and then go to the event table in the **Properties** window. Scroll down to the MouseEnter event and select the event name.
2. Press DELETE to delete the event handler name, and then press ENTER. The IDE automatically disconnects the event handler from the panel. The walls are still connected, but now you can move your mouse outside of the maze to get to the **Finish** label at the bottom.
3. Save and run your program, and be sure the **Finish** label plays the sound, shows the message box, and closes the game. After you're sure it works, enable the panel's MouseEnter event handler by selecting it, going to the event table in the **Properties** window, scrolling down to the MouseEnter line, and selecting**wall\_MouseEnter** from the drop-down list.

To try other features

* Replace the sounds in the game with sounds you like better.
* Set it up so that the game only plays a sound when the mouse pointer hits a wall, but doesn't play a sound when the program starts.
* Instead of closing the program when the player wins, have the pointer move back to the starting point.
* Change some of the wall colors and make the game play different sounds with different wall colors.

To continue or review