KHARKIV NATIONAL UNIVERSITY OF RADIO ELECTRONICS

INFORMATICS LABORATORY WORK #4

MAZE GAME CREATION

Associate Professor A.S. Eremenko,

Associate Professor A.V. Persikov

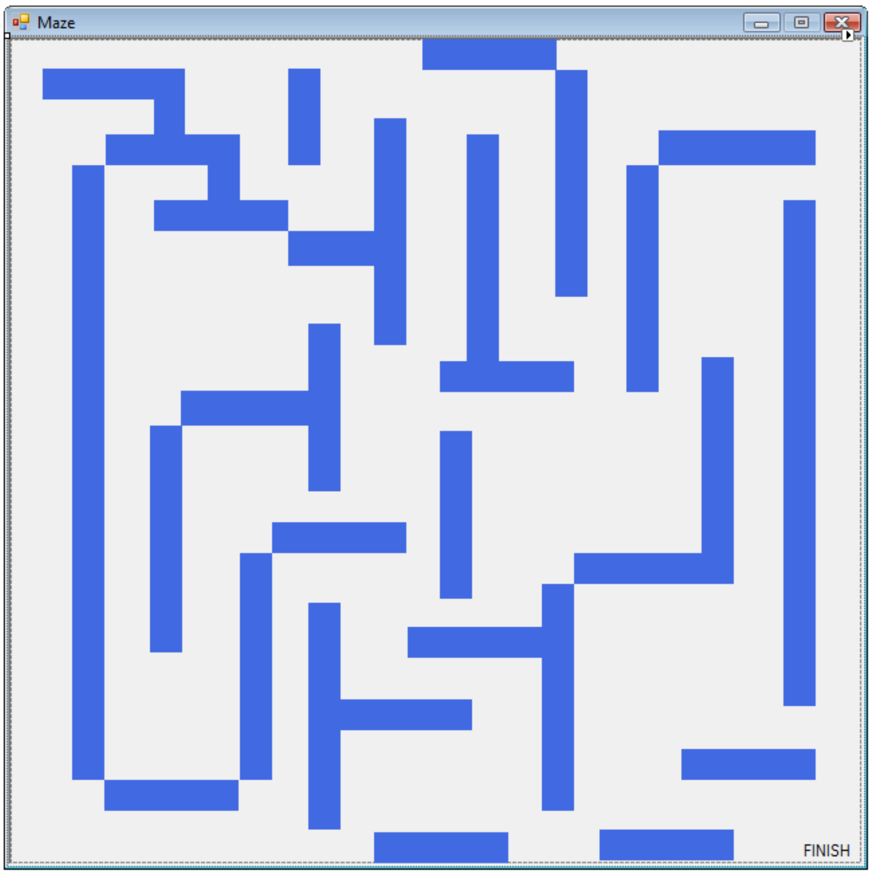
Maze

In this lab, you build a maze game, where the user has to move the mouse pointer from the start to the finish without touching any of the walls. You learn how to:

- lay out a form using a Panel container;
- build a maze using Label controls;
- write code to show a message box;
- set up event handlers for mouse events;
- play sounds in your program;
- organize your code using classes.

Here's how the maze will work: The mouse pointer starts at the upper-left corner of the maze. The user navigates through the maze, being careful not to touch any of the walls with the pointer. If the pointer touches one of the walls, it automatically jumps back to the start. But if the pointer reaches the Finish label at the end of the maze, a "Congratulations" message box opens, and the game ends.

When you finish, your program will look like the following picture.

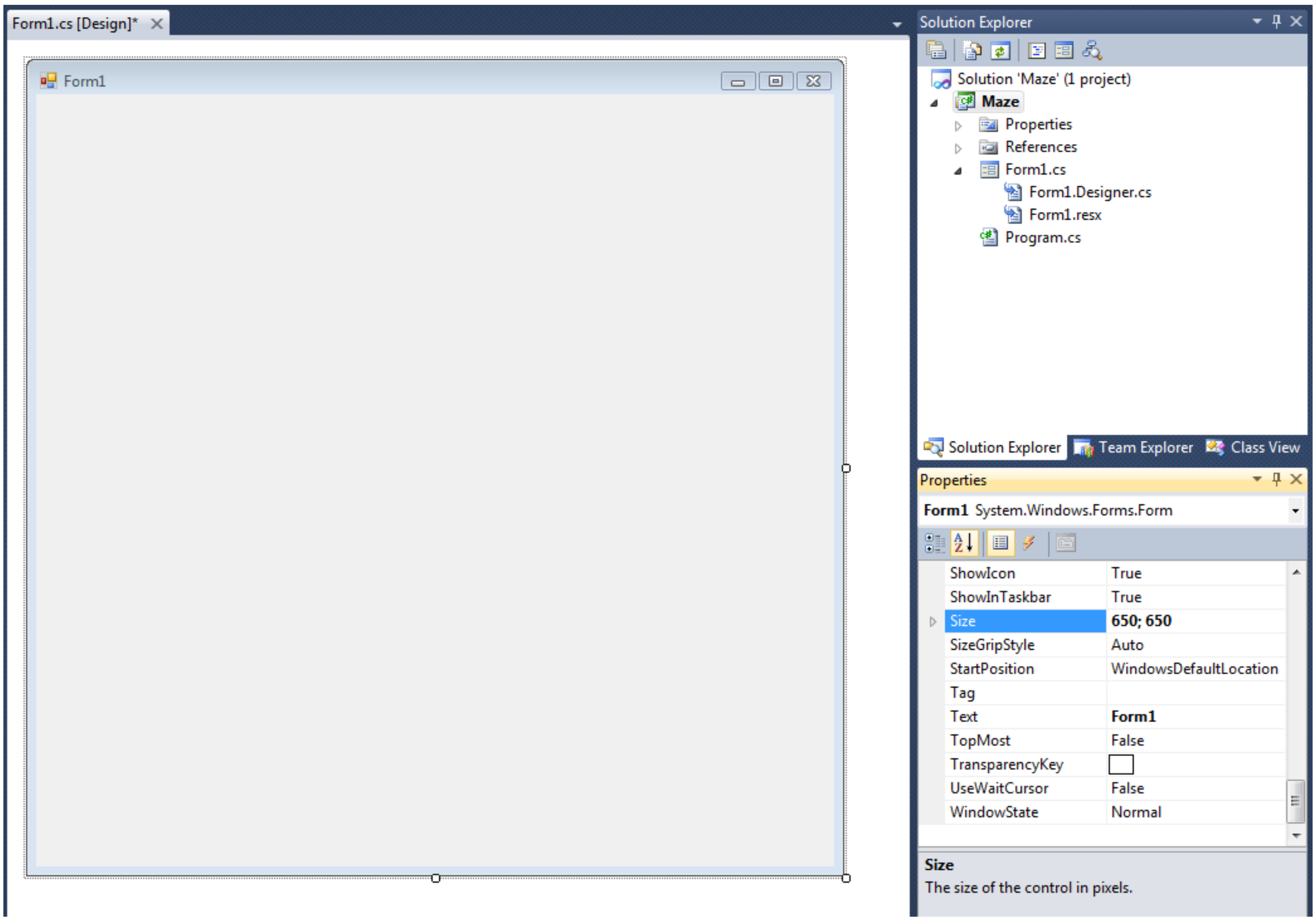


Step 1: Create a Project and Add a Panel to Your Form

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 C#.
3. Click the **Windows Forms Application** icon, and then type Maze as the name.
4. Set the form properties:
a. 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.



```
Solution 'Maze' (1 project)
  Maze
    Properties
    References
    Form1.cs
      Form1.Designer.cs
      Form1.resx
    Program.cs
```

Form1 System.Windows.Forms.Form

ShowIcon	True
ShowInTaskbar	True
Size	650; 650
SizeGripStyle	Auto
StartPosition	WindowsDefaultLocation
Tag	
Text	Form1
TopMost	False
TransparencyKey	☐
UseWaitCursor	False
WindowState	Normal

Size
The size of the control in pixels.

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

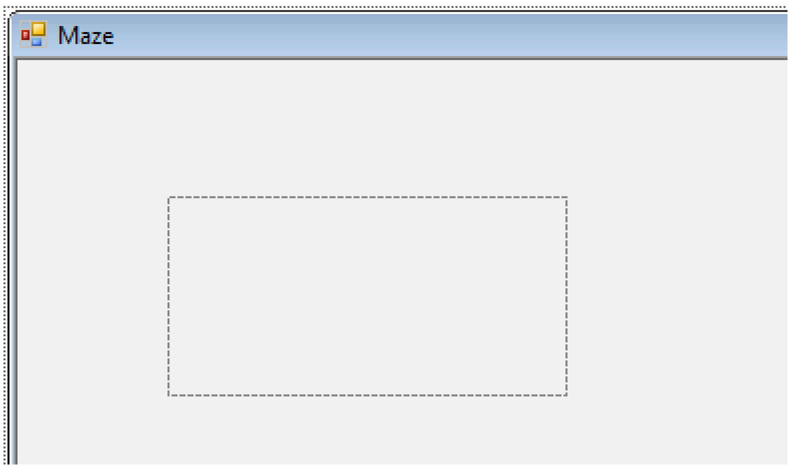
Now you have a form that's a fixed size, which the user can't maximize.

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.

5. 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

6. 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.



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

7. 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**.

8. Save the project by clicking the **Save All** toolbar button, which appears as follows.

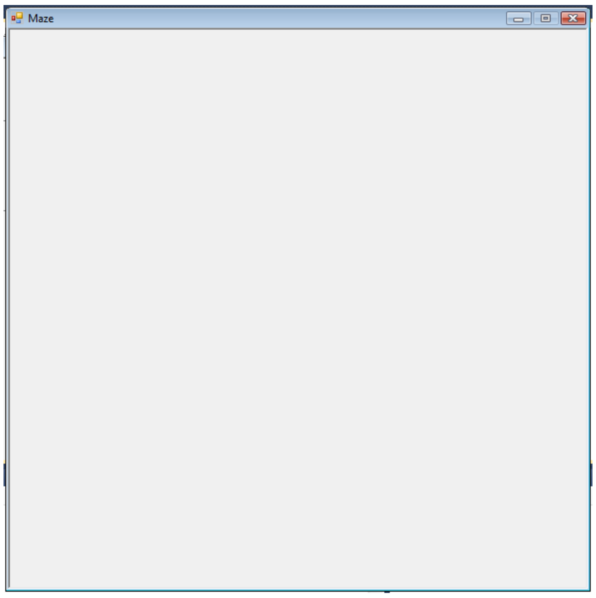
Save All button

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

Start Debugging toolbar button

▶

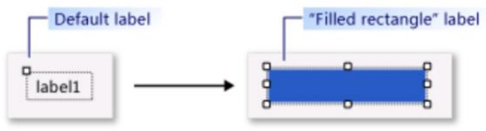
When running, your form should look like the following picture.



10. 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.)

Step 2: 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.



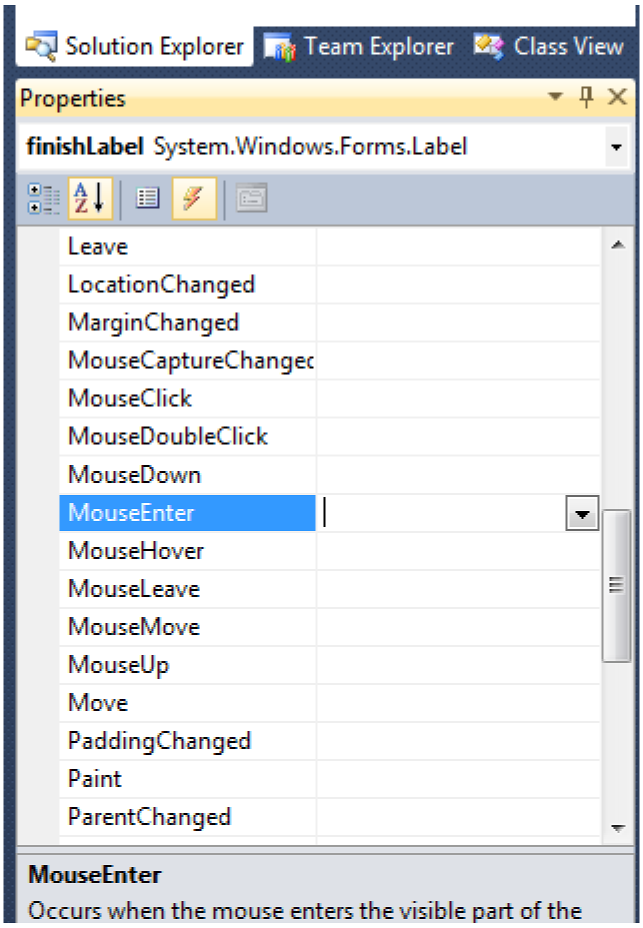
Your Label control should now be a filled rectangle.

3. 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.

4. 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.
5. 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**.
6. Drag your new **Finish** label to the end of the maze. That's the target that the user needs to hit.
7. Save your project and run your program again. The following is an example of a finished maze form. (Your maze will look different.)

Step 3: 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.



finishLabel System.Windows.Forms.Label

Leave	
LocationChanged	
MarginChanged	
MouseCaptureChangec	
MouseClick	
MouseDoubleClick	
MouseDown	
MouseEnter	
MouseHover	
MouseLeave	
MouseMove	
MouseUp	
Move	
PaddingChanged	
Paint	
ParentChanged	

MouseEnter
Occurs when the mouse enters the visible part of the

Event icon

Property icon

MouseEnter event

2. 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.

```
private void finishLabel_MouseEnter(object sender, EventArgs e)
{

}
```

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

3. 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.

```
private void finishLabel_MouseEnter(object sender, EventArgs e)
{
 MessageBox.Show("Congratulations");
 Close();
}
```

4. 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.

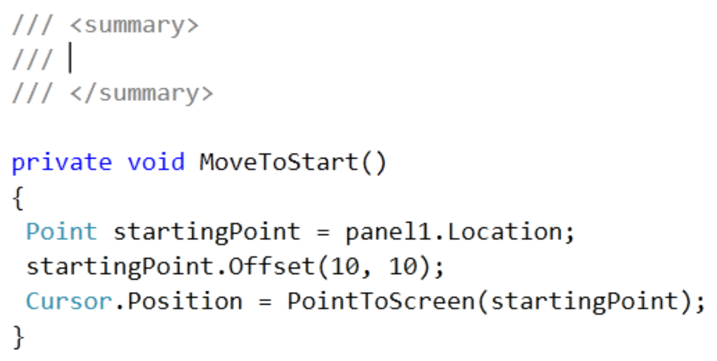
5. Save and run your program. Move your mouse pointer over the **Finish** label. It should open the message, and then close the program.

Step 4: 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.

```
private void MoveToStart()
{
 Point startingPoint = panel1.Location;
 startingPoint.Offset(10, 10);
 Cursor.Position = PointToScreen(startingPoint);
}
```

3. 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.



```
/// <summary>
/// |
/// </summary>

private void MoveToStart()
{
 Point startingPoint = panel1.Location;
 startingPoint.Offset(10, 10);
 Cursor.Position = PointToScreen(startingPoint);
}
```

```
/// <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);
}
```

4. 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.)

5. 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.

```
public Form1()
{
 InitializeComponent();
 MoveToStart();
}
```

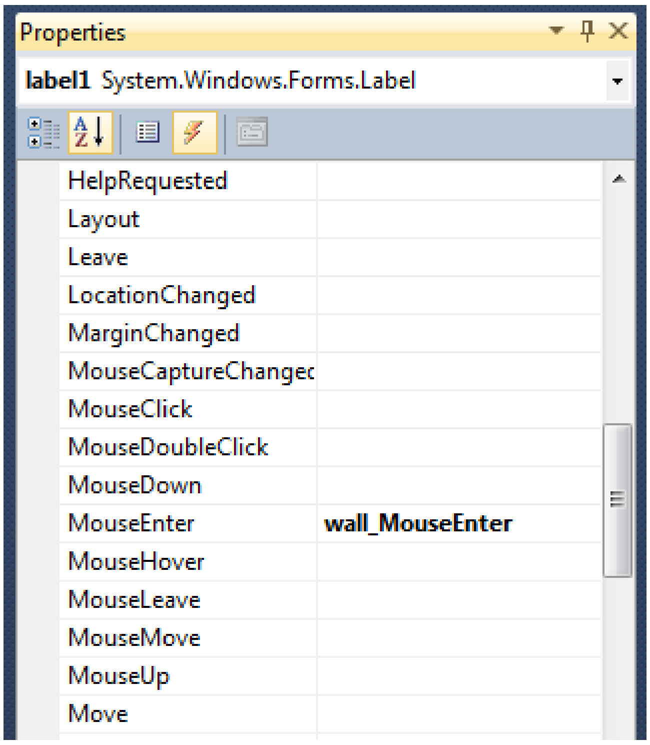
Note the call to the **MoveToStart()** method underneath **InitializeComponent()**. If you're programming in Visual C#, remember to end that line with a semicolon (;), or your program won't build.

7. 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.

Step 5: 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.

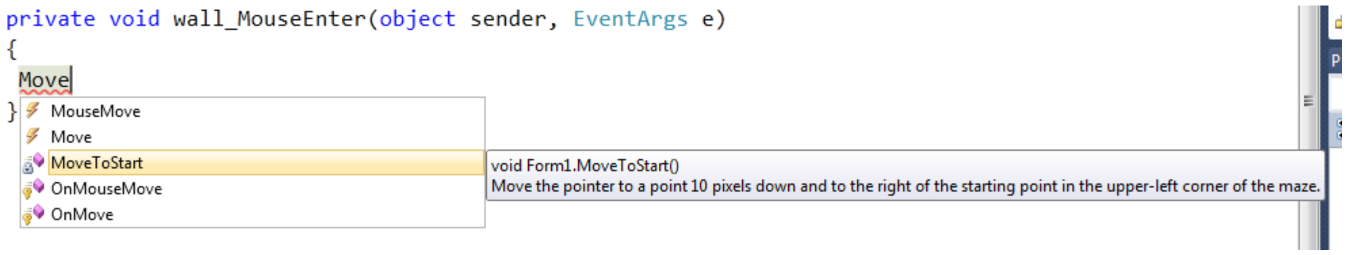


HelpRequested	
Layout	
Leave	
LocationChanged	
MarginChanged	
MouseCaptureChangec	
MouseClick	
MouseDoubleClick	
MouseDown	
MouseEnter	wall_MouseEnter
MouseHover	
MouseLeave	
MouseMove	
MouseUp	
Move	

```
private void wall_MouseEnter(object sender, EventArgs e)
{

}
```

4. 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.



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

void Form1.MoveToStart()
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.

5. 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.

```
private void wall_MouseEnter(object sender, EventArgs e)
{
 // When the mouse pointer hits a wall or enters the panel,
 // call the MoveToStart() method.

 MoveToStart();
}
```

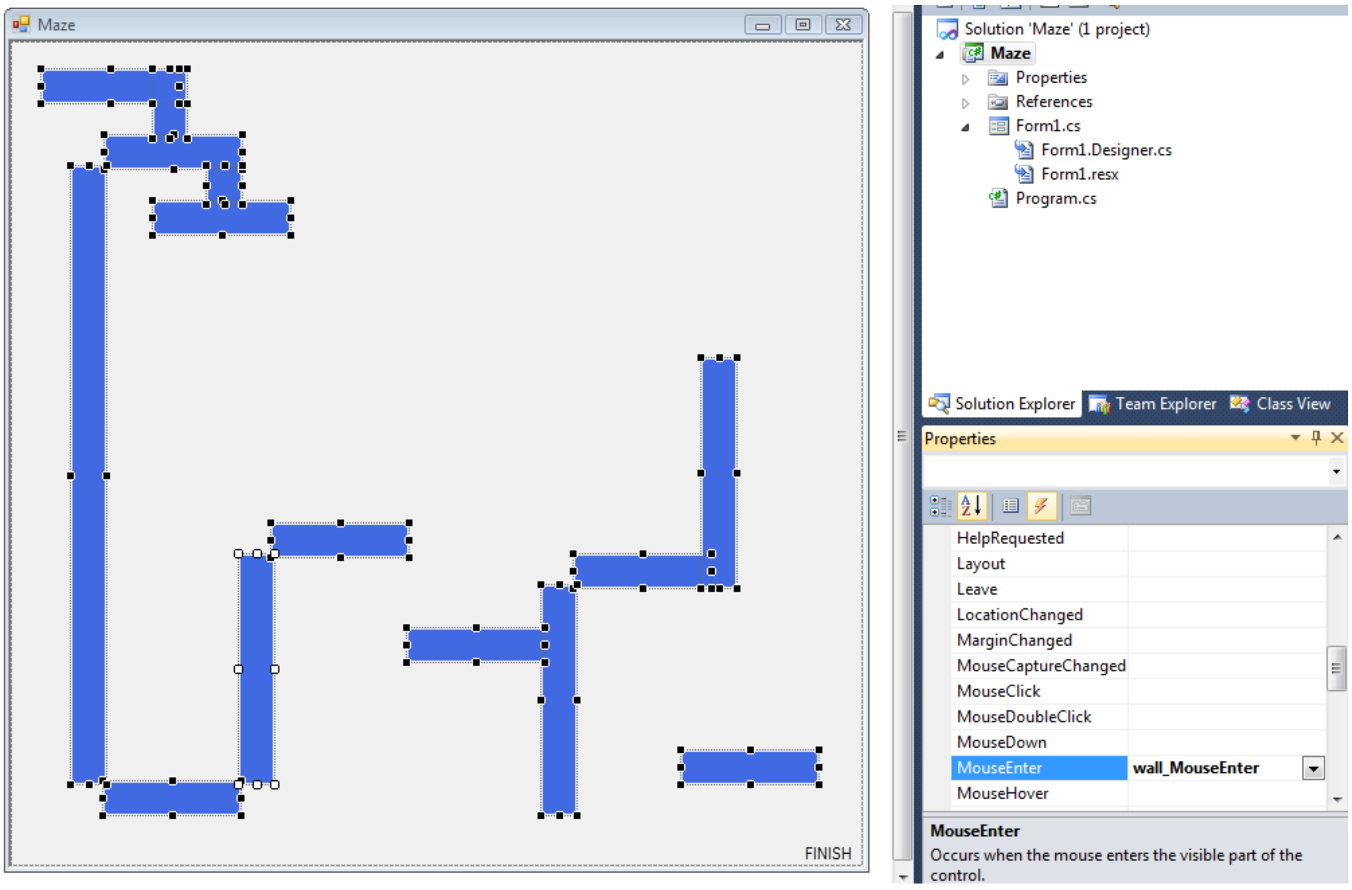
6. Save and run your program. Move your mouse pointer over the wall that you connected the event handler to. (If you don't remember which one you chose, move your mouse pointer over each wall until you find the right one.) As soon as you touch it, it should send your mouse pointer back to the start.

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.

7. Go to Windows Forms Designer, and from the **Edit** menu, click **Select All**.

8. 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.

9. 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.



MouseEnter wall_MouseEnter

MouseEnter
Occurs when the mouse enters the visible part of the control.

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.

10. 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.)

11. 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.

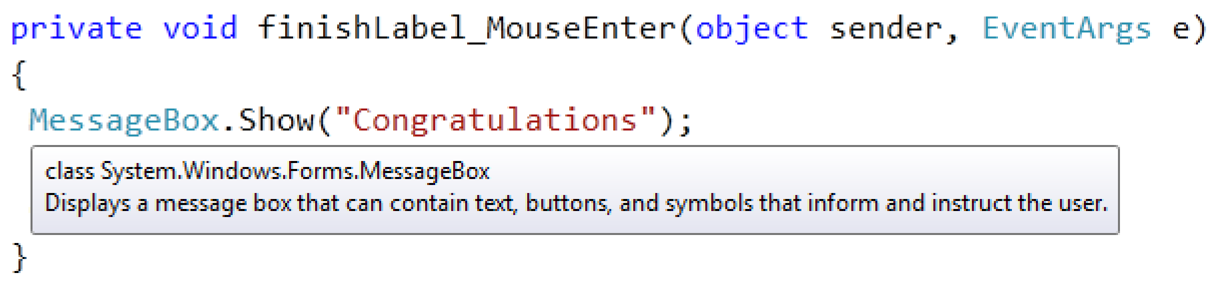
Step 6: Add a SoundPlayer

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

```
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();
 }
```

2. 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.



```
private void finishLabel_MouseEnter(object sender, EventArgs e)
{
 MessageBox.Show("Congratulations");
 class System.Windows.Forms.MessageBox
 Displays a message box that can contain text, buttons, and symbols that inform and instruct the user.
}
```

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.)

3. 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.)

Step 7: Add Code to Your Form 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.

```
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();
     }
```

2. 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.

```
private void MoveToStart()
{
 startSoundPlayer.Play();
 Point startingPoint = panel1.Location;
 startingPoint.Offset(10, 10);
 Cursor.Position = PointToScreen(startingPoint);
}
```

3. 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.

Step 8: Run Your Program and Try Other Features

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.
5. 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.
6. 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.
7. 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.