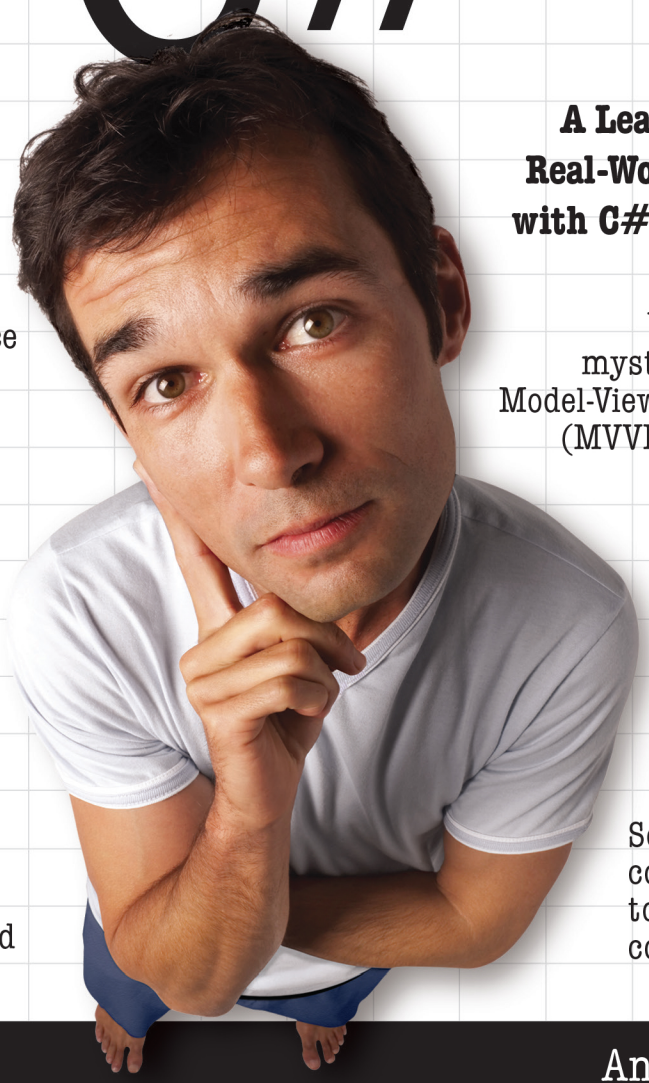
A Brain-Friendly Guide

3rd Edition
Includes a Bonus Windows Phone Project

Head First C#

Boss your objects around with abstraction and inheritance



A Learner's Guide to Real-World Programming with C#, XAML, and .NET

Unravel the mysteries of the Model-View-ViewModel (MVVM) pattern

Build a fully functional retro classic arcade game

See how Jimmy used collections and LINQ to wrangle an unruly comic book collection

Learn how asynchronous programming helped Sue keep her users thrilled

O'REILLY®

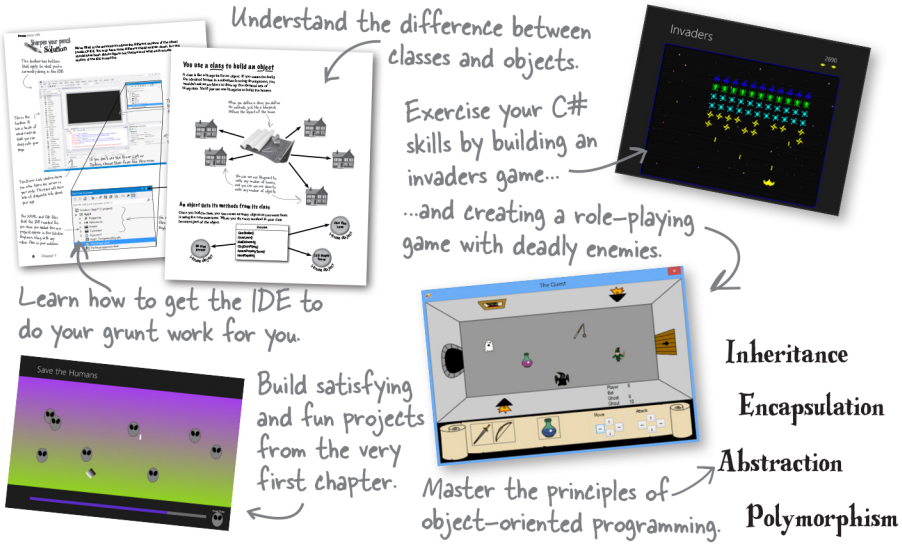
Andrew Stellman
& Jennifer Greene

Head First C#

Programming/C#/.NET

What will you learn from this book?

Head First C# is a complete learning experience for programming with C#, XAML, the .NET Framework, and Visual Studio. Built for your brain, this book keeps you engaged from the first chapter, where you'll build a fully functional video game. After that, you'll learn about classes and object-oriented programming, draw graphics and animation, query your data with LINQ, and serialize it to files. And you'll do it all by building games, solving puzzles, and doing hands-on projects. By the time you're done you'll be a solid C# programmer, and you'll have a great time along the way!



Why does this book look so different?

We think your time is too valuable to spend struggling with new concepts. Using the latest research in cognitive science and learning theory to craft a multi-sensory learning experience, *Head First C#* uses a visually rich format designed for the way your brain works, not a text-heavy approach that puts you to sleep.

"If you want to learn C# in depth and have fun doing it, this is THE book for you."

—Andy Parker, fledgling C# programmer

"*Head First C#* will guide beginners of all sorts to a long and productive relationship with C# and the .NET Framework."

—Chris Burrows, Developer on Microsoft's C# Compiler team

"*Head First C#* got me up to speed in no time for my first large scale C# development project at work—I highly recommend it."

—Shalewa Odusanya, Technical Account Manager, Google

US $49.99 CAN $52.99

ISBN: 978-1-449-34350-7

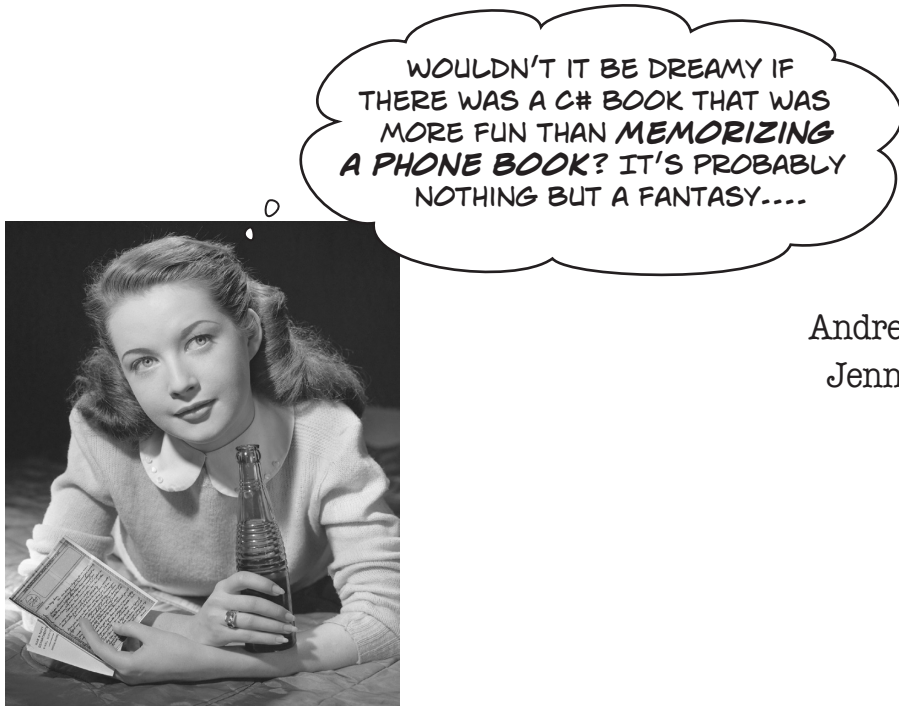
twitter.com/headfirstlabs
facebook.com/HeadFirst

O'REILLY®

oreilly.com
headfirstlabs.com

Head First C#

Third Edition



Andrew Stellman
Jennifer Greene

O'REILLY®

Beijing • Cambridge • Köln • Sebastopol • Tokyo

Head First C#

Third Edition

by Andrew Stellman and Jennifer Greene

Copyright © 2013 Andrew Stellman and Jennifer Greene. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Series Creators: Kathy Sierra, Bert Bates

Cover Designers: Louise Barr, Karen Montgomery

Production Editor: Melanie Yarbrough

Proofreader: Rachel Monaghan

Indexer: Ellen Troutman-Zaig

Page Viewers: Quentin the whippet and Tequila the pomeranian

Printing History:

November 2007: First Edition.
May 2010: Second Edition.
August 2013: Third Edition.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. The *Head First* series designations, *Head First C#*, and related trade dress are trademarks of O'Reilly Media, Inc.

Microsoft, Windows, Visual Studio, MSDN, the .NET logo, Visual Basic and Visual C# are registered trademarks of Microsoft Corporation.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

No bees, space aliens, or comic book heroes were harmed in the making of this book.

ISBN: 978-1-449-34350-7

[M]

17 bonus project!

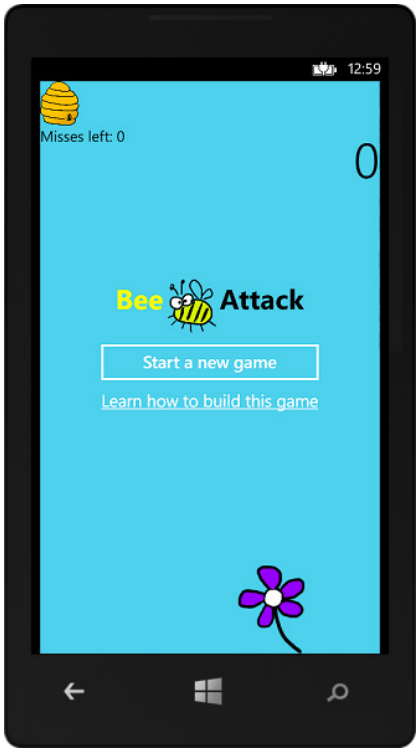
Build a Windows Phone app

You're already able to write Windows Phone apps.

Classes, objects, XAML, encapsulation, inheritance, polymorphism, LINQ, MVVM… you've got all of the tools you need to build great Windows Store apps and desktop apps. But did you know that you can **use these same tools to build apps for *Windows Phone***? It's true! In this bonus project, we'll walk you through creating a game for Windows Phone. And if you don't have a Windows Phone, don't worry—you'll still be able to use the **Windows Phone emulator** to play it. Let's get started!

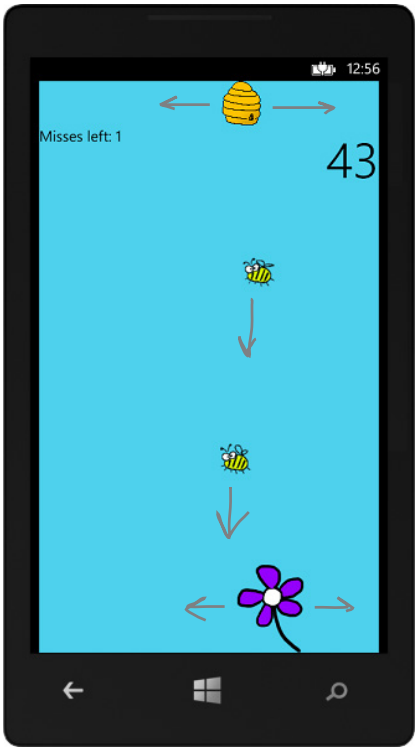
Bee Attack!

You'll be building a Windows Phone 8 game called **Bee Attack**. There's a hive of seriously ticked-off bees, and the only way you'll pacify them is with a tasty, tasty flower. The more bees you catch with the flower, the higher your score.



Catch the bees as they fly down the screen.

The hive moves back and forth across the top of the screen, launching bees down at your flower. As you catch more and more bees, the hive shoots them out more frequently and moves farther to the left or right between bees.



Control the flower with touch gestures.

You control the flower by dragging left and right across the screen. It will follow your finger as you drag, letting you catch the bees as they fly down toward the bottom. You've only got five misses before the game ends, and it gets harder as the game goes on.

Before you begin...

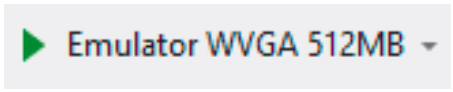
To build this project, you need to **install Visual Studio 2012 for Windows Phone**. You can download the free Express edition here:

http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-for-windows-phone

You have two options for actually running the game. The easiest way to do it is to use the **Windows Phone Emulator** that ships as part of Visual Studio.

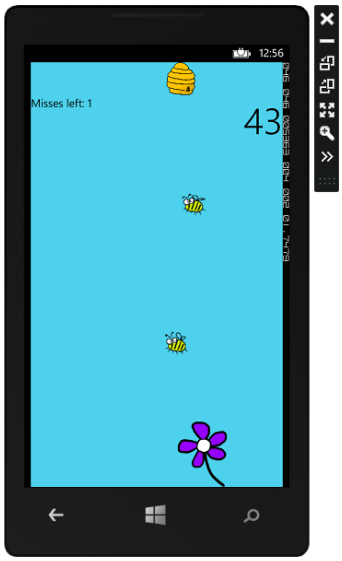
Run the game in the emulator.

By default, Visual Studio for Windows Phone runs its apps in the emulator, which emulates a complete Windows Phone 8 (including Internet connection, a fake GSM network, and more).



The Run button in the IDE launches the emulator.

The emulator lets you run your Windows Phone apps on your computer.



Watch it!

The Windows Phone Emulator requires Hyper-V

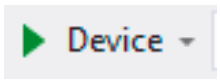
Hyper-V is the virtualization technology for Windows 8, but not every CPU or edition of Windows 8 can run Hyper-V. Microsoft has this guide to help you get it up and running:

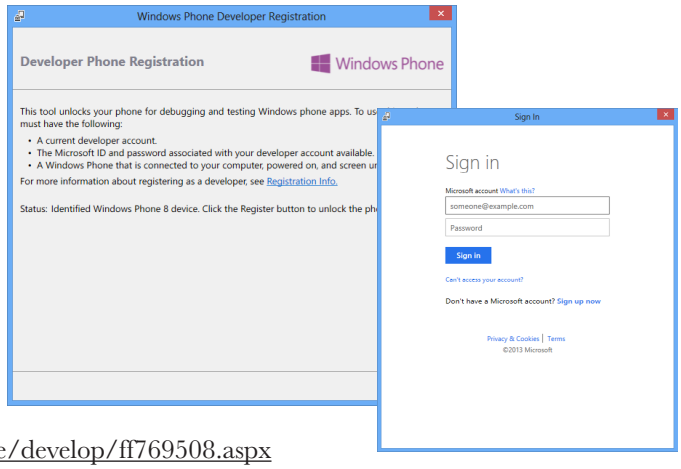
http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj863509.aspx

*Hyper-V **will** run on a virtual machine hosted inside VMWare, VirtualBox, Parallels, or another virtualization product if your processor supports it. You'll need to consult your VM software documentation or support website for details (you may need to enable an option like "nested virtual machines" or "virtualize VT-x or AMD-V" to make it work).*

Run the game on your Windows Phone.

If you connect a Windows Phone 8 to your computer via a USB cable, then you can debug your app on the device.





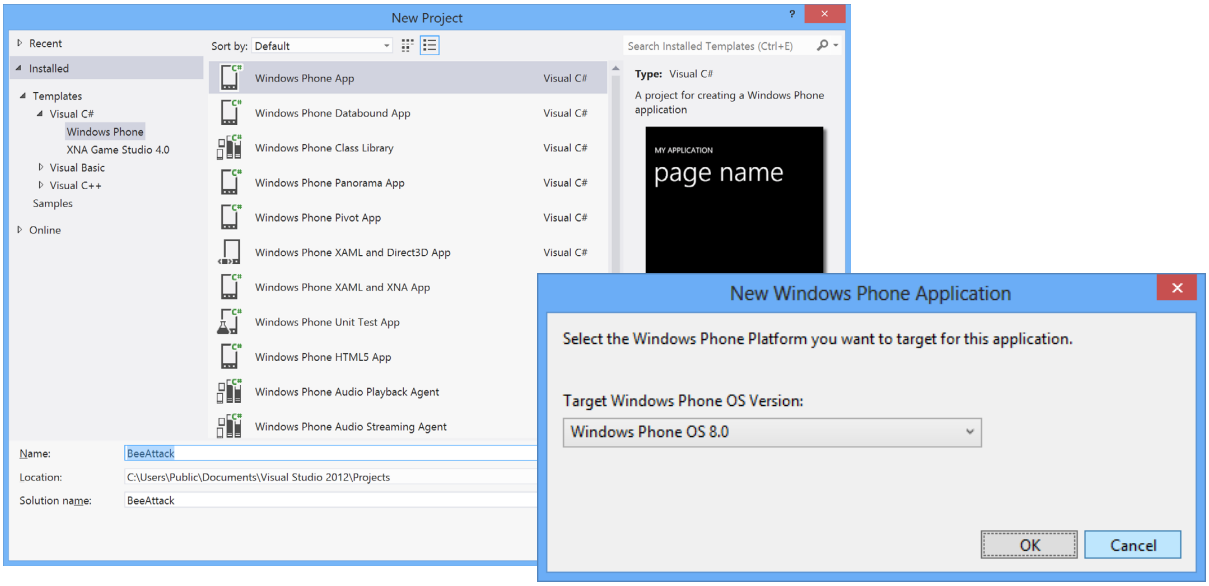
For this to work, your phone must be registered with an active Windows Phone Dev Center account (which might cost money): **https://dev.windowsphone.com/.**

Once you have this account set up, you can register your phone for development. This page will show you how:

http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769508.aspx

1 CREATE A NEW WINDOWS PHONE APP.

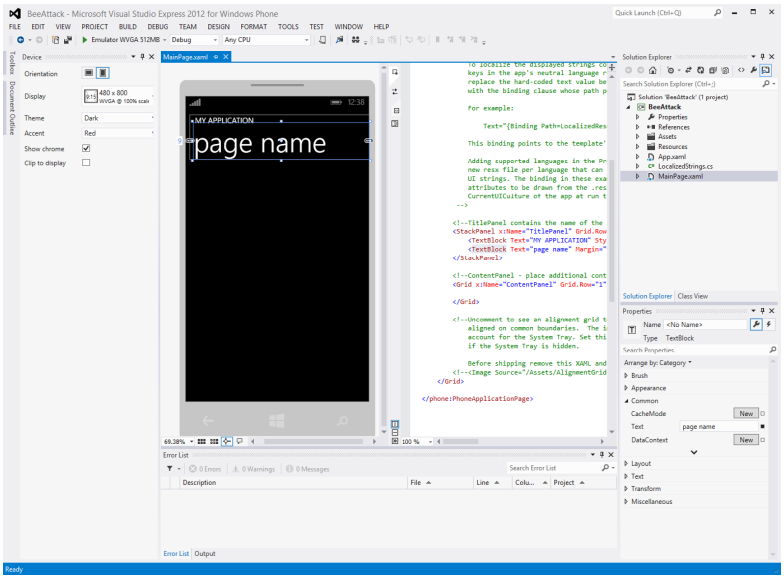
Open Visual Studio 2012 for Windows Phone and **create a new Windows Phone App project called BeeAttack**. (If you choose a different name, that's OK—but the namespaces in our screenshots will be a little different from yours.)



2 LOOK THROUGH THE FILES THAT THE IDE CREATED.

The Visual Studio for Windows Phone IDE should be very familiar, because it's almost identical to the IDE for the Windows 8 and Desktop editions. When you create the new project, the IDE automatically adds files that include:

- The XAML and C# files for the main page (*MainPage.xaml* and *MainPage.xaml.cs*)
- The main app file (*App.xaml* and *App.xaml.cs*)
- An *Assets* folder
- A C# source file to hold static resources for localized strings (*LocalizedStrings.cs*)
- An app manifest, resources, and a few more files and folders (but nothing else you'll need for this project)



3 HIDE THE TITLE PANEL ON THE MAIN PAGE.

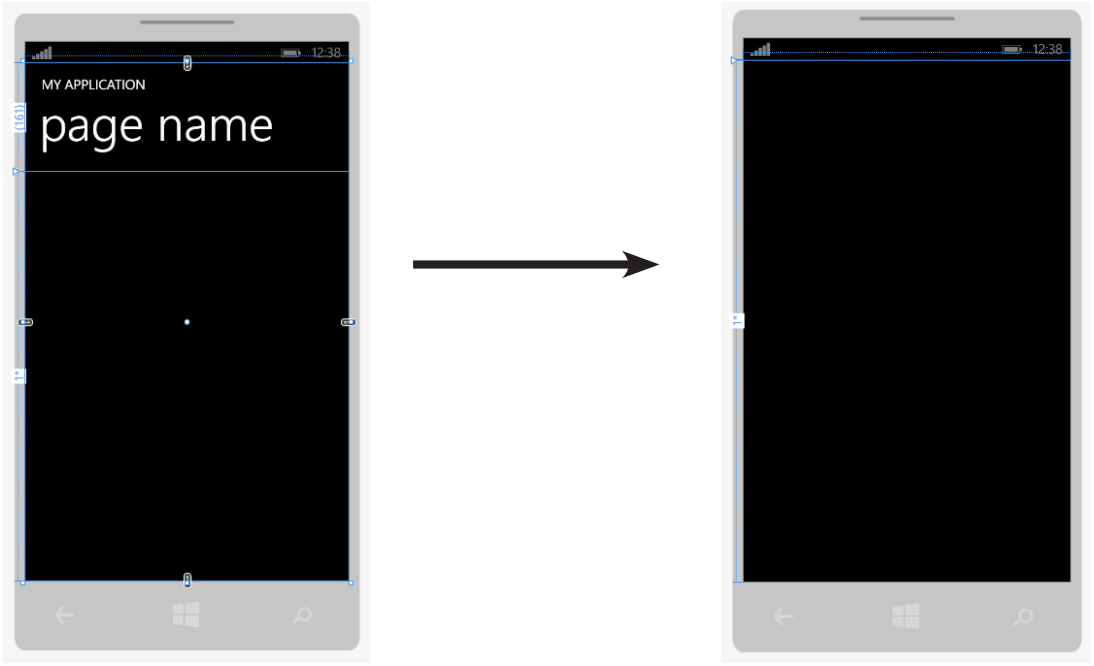
The main page, *MainPage.xaml*, should already be open in the IDE (if it's not, open it). This is the main page displayed on your app. Look through the XAML code for it and find the StackPanel called TitlePanel.

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock Text="MY APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>
    <TextBlock Text="page name" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

Edit the XAML and **add `Visibility="Collapsed"` to the StackPanel** so the title TextBlocks disappear. This will make your app appear to be full screen.

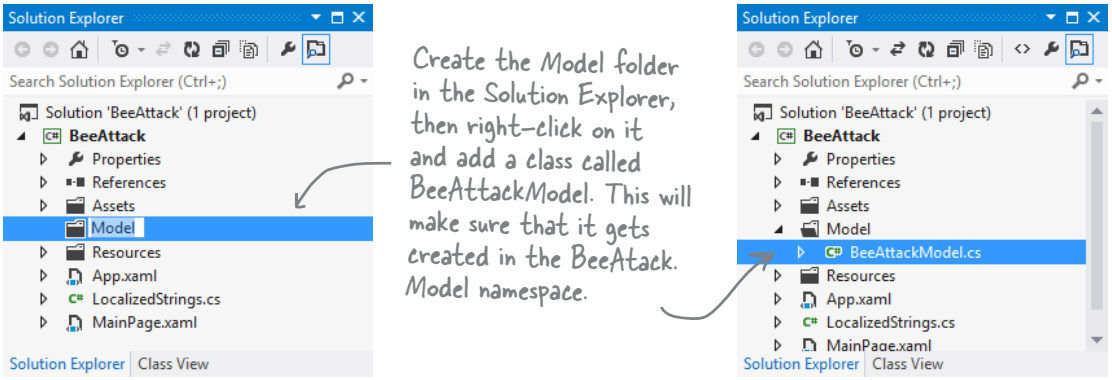
The IDE created MainPage.xaml with a StackPanel named TitlePanel that contains two TextBlocks. Add this Visibility property to make it disappear.

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28" Visibility="Collapsed">
    <TextBlock Text="MY APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>
    <TextBlock Text="page name" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```



4 CREATE THE MODEL FOLDER AND ADD THE BEEATTACKMODEL CLASS.

BeeAttack is an MVVM app, so it needs its Model, View, and ViewModel layers. We'll start by creating the Model layer. In this case, the Model is a single class called `BeeAttackModel`, which lives in the *Model* folder and namespace. Create the *Model* folder and add the `BeeAttackModel` class to it:



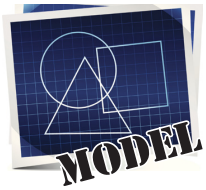
Here's the code for the `BeeAttackModel` class. It has properties to get the number of misses left, the score, and the time between launching bees, and it has a method to start the game. It also has methods to tell it the player moved the flower, tell it a bee landed, and get the hive location for the next bee launch.

```
using Windows.Foundation;

class BeeAttackModel {
    public int MissesLeft { get; private set; }
    public int Score { get; private set; }
    public TimeSpan TimeBetweenBees  {
        get {
            double milliseconds = 500;
            milliseconds = Math.Max(milliseconds - Score * 2.5, 100);
            return TimeSpan.FromMilliseconds(milliseconds);
        }
    }

    private double _flowerWidth;
    private double _beeWidth;
    private double _flowerLeft;
    private double _playAreaWidth;
    private double _hiveWidth;
    private double _lastHiveLocation;
    private bool _gameOver;
    private readonly Random _random = new Random();

    public void StartGame(double flowerWidth, double beeWidth, double playAreaWidth, double hiveWidth) {
        _flowerWidth = flowerWidth;
        _beeWidth = beeWidth;
        _playAreaWidth = playAreaWidth;
        _hiveWidth = hiveWidth;
        _lastHiveLocation = playAreaWidth / 2;
        MissesLeft = 5;
        Score = 0;
        _gameOver = false;
        OnPlayerScored();
    }
```

The ViewModel will use these properties to update the score and number of misses left.

The game speeds up as the player catches more and more bees. This property calculates the time between bees based on the score.

The StartGame() method resets the game. The ViewModel will pass it the widths of the flower, bee, play area, and hive, which it uses in its methods.



This method is called any time the player moves the flower. It just updates the flower's position.

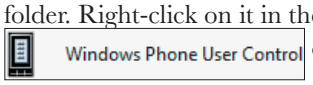
```
    public void MoveFlower(double flowerLeft) {
        _flowerLeft = flowerLeft;
    }

    public void BeeLanded(double beeLeft) {
        if ((beeLeft < _flowerLeft) || (beeLeft > _flowerLeft + _flowerWidth)) {
            if (MissesLeft > 0) {
                MissesLeft--;
                OnMissed();
            } else {
                _gameOver = true;
                OnGameOver();
            }
        }
        else if (!_gameOver) {
            Score++;
            OnPlayerScored();
        }
    }

    public double NextHiveLocation() {
        double delta = 10 + Math.Max(1, Score * .5);

        if (_lastHiveLocation <= _hiveWidth * 2)
            _lastHiveLocation += delta;
        else if (_lastHiveLocation >= _playAreaWidth - _hiveWidth * 2)
            _lastHiveLocation -= delta;
        else
            _lastHiveLocation += delta * (_random.Next(2) == 0 ? 1 : -1);


        return _lastHiveLocation;
    }

    public EventHandler Missed;
    private void OnMissed() {
        EventHandler missed = Missed;
        if (missed != null)
            missed(this, new EventArgs());
    }

    public EventHandler GameOver;
    private void OnGameOver() {
        EventHandler gameOver = GameOver;
        if (gameOver != null)
            gameOver(this, new EventArgs());
    }

    public EventHandler PlayerScored;
    private void OnPlayerScored() {
        EventHandler playerScored = PlayerScored;
        if (playerScored != null)
            playerScored(this, new EventArgs());
    }
}
```

Wherever a bee lands, the Model checks if the bee's left corner is inside the boundaries of the flower. If it's not, the player missed; otherwise, the player scored.

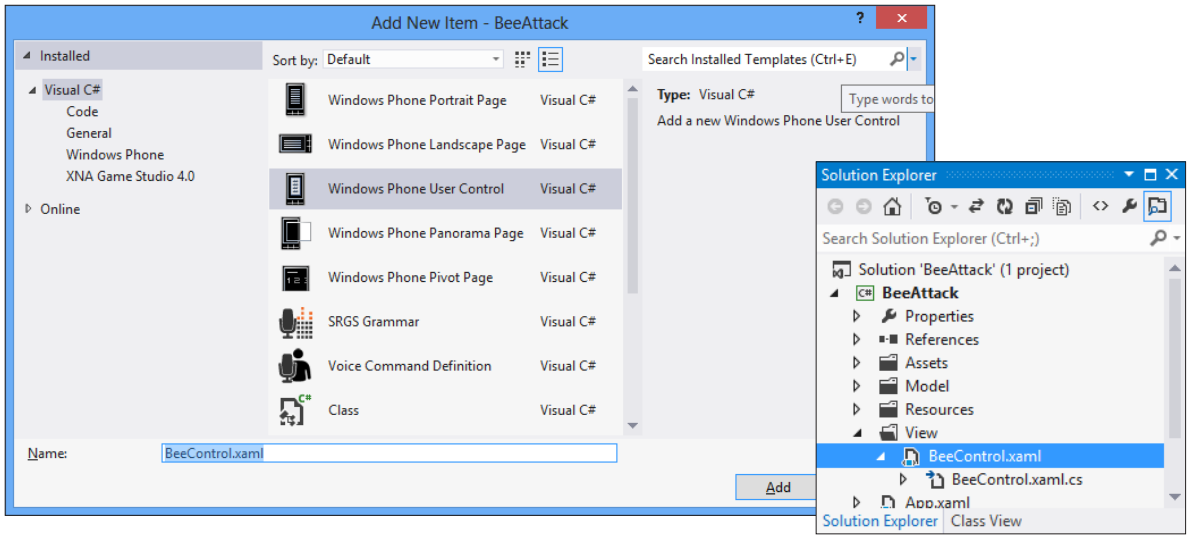
Every time a bee is launched, the ViewModel uses this method to find the next horizontal location for the hive to launch the bee. The method makes sure the hive doesn't move too far—as the player scores more, the hive moves further between bees.

The ViewModel listens to these three events so it can update the View whenever the player misses, scores, or loses the game.

5 CREATE THE VIEW FOLDER AND BUILD THE BeeControl.

The View consists of two Windows Phone user controls. The first one is called `BeeControl`, an animated bee picture that moves down the play area. Start by creating the *View* folder. Right-click on it in the Solution Explorer and add a new item, then **add a new** 

called *BeeControl.xaml* to the folder:



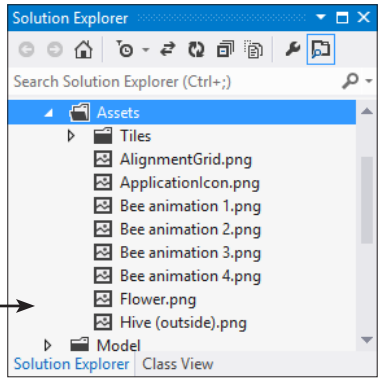
Edit the XAML code in *BeeControl.xaml*. Find the Grid named LayoutRoot, give it a transparent background, and add an Image control to it named `image`. Here's what the XAML should look like:

```
<Grid x:Name="LayoutRoot" Background="Transparent">

    <Image x:Name="image" Stretch="Fill"/>

</Grid>
```

You'll need the animated flapping bee graphics, as well as the flower and hive graphics. **Download the graphics files from the Head First Labs website:**

http://www.headfirstlabs.com/hfcsharp

Then right-click on the *Assets* folder and **add the files as existing items**. Once you've added the files, your *Assets* folder should look like this:



Now you're ready to **add the C# code-behind for *BeeControl***. Here's the code for it, which should be added to *BeeControl.xaml.cs*:

```
using Microsoft.Phone.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Imaging;

public sealed partial class BeeControl : UserControl {
    public readonly Storyboard FallingStoryboard;

    public BeeControl() {
        this.InitializeComponent();
        StartFlapping(TimeSpan.FromMilliseconds(30));
    }

    public BeeControl(double X, double fromY, double toY, EventHandler completed) : this() {
        FallingStoryboard = new Storyboard();
        DoubleAnimation animation = new DoubleAnimation();

        Storyboard.SetTarget(animation, this);
        Canvas.SetLeft(this, X);
        Storyboard.SetTargetProperty(animation, new PropertyPath("(Canvas.Top)"));
        animation.From = fromY;
        animation.To = toY;
        animation.Duration = TimeSpan.FromSeconds(1);

        if (completed != null) FallingStoryboard.Completed += completed;

        FallingStoryboard.Children.Add(animation);
        FallingStoryboard.Begin();
    }

    public void StartFlapping(TimeSpan interval) {
        List<string> imageNames = new List<string>() {
            "Bee animation 1.png", "Bee animation 2.png", "Bee animation 3.png", "Bee animation 4.png"
        };

        Storyboard storyboard = new Storyboard();
        ObjectAnimationUsingKeyFrames animation = new ObjectAnimationUsingKeyFrames();
        Storyboard.SetTarget(animation, image);
        Storyboard.SetTargetProperty(animation, new PropertyPath("Source"));

        TimeSpan currentInteval = TimeSpan.FromMilliseconds(0);
        foreach (string imageName in imageNames) {
            ObjectKeyFrame keyFrame = new DiscreteObjectKeyFrame();
            keyFrame.Value = CreateImageFromAssets(imageName);
            keyFrame.KeyTime = currentInteval;
            animation.KeyFrames.Add(keyFrame);
            currentInteval = currentInteval.Add(interval);
        }

        storyboard.RepeatBehavior = RepeatBehavior.Forever;
        storyboard.AutoReverse = true;
        storyboard.Children.Add(animation);
        storyboard.Begin();
    }

    private static BitmapImage CreateImageFromAssets(string imageFilename) {
        return new BitmapImage(new Uri("/Assets/" + imageFilename, UriKind.RelativeOrAbsolute));
    }
}
```

After a storyboard completes its animation, it fires its `Completed` event. This overloaded `BeeControl` constructor takes an `EventHandler` delegate as a parameter, which the ViewModel uses to figure out when the bee lands.

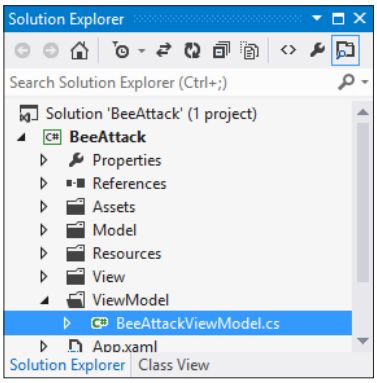
This animation moves the bee down the Canvas play area, starting at the hive and ending at the flower.

The EventHandler delegate passed in as an argument is hooked up to the Storyboard's Completed event.

This key frame animation flips through the animation frames to make the bee's wings flap.

6 CREATE THE VIEWMODEL.

The ViewModel layer consists of a single class, `BeeAttackViewModel`, which uses the methods, properties, and events in the Model to update the controls in the View. **Create the *ViewModel* folder and add a `BeeAttackViewModel` class.** Here's the code for it:



The View uses an ItemsPanel to bind this collection of BeeControl objects to the Children of a Canvas, so the ViewModel can add bees by updating its _beeControls field.

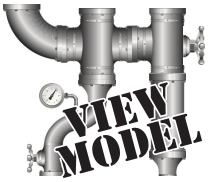
```
using View;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Threading;

class BeeAttackViewModel : INotifyPropertyChanged {
    public INotifyCollectionChanged BeeControls { get { return _beeControls; } }
    private readonly ObservableCollection<BeeControl> _beeControls
                                                    = new ObservableCollection<BeeControl>();

    public Thickness FlowerMargin { get; private set; }
    public Thickness HiveMargin { get; private set; }
    public int MissesLeft { get { return _model.MissesLeft; } }
    public int Score { get { return _model.Score; } }
    public Visibility GameOver { get; private set; }

    private Size _beeSize;
    private readonly Model.BeeAttackModel _model = new Model.BeeAttackModel();
    private readonly DispatcherTimer _timer = new DispatcherTimer();
    private double _lastX;
    private Size _playAreaSize { get; set; }
    private Size _hiveSize { get; set; }
    private Size _flowerSize { get; set; }

    public BeeAttackViewModel() {
        _model.Missed += MissedEventHandler;
        _model.GameOver += GameOverEventHandler;
        _model.PlayerScored += PlayerScoredEventHandler;

        _timer.Tick += HiveTimerTick;

        GameOver = Visibility.Visible;
        OnPropertyChanged("GameOver");
    }

    public void StartGame(Size flowerSize, Size hiveSize, Size playAreaSize) {
        _flowerSize = flowerSize;
        _hiveSize = hiveSize;
        _playAreaSize = playAreaSize;
        _beeSize = new Size(playAreaSize.Width / 10, playAreaSize.Width / 10);
        _model.StartGame(flowerSize.Width, _beeSize.Width, playAreaSize.Width, hiveSize.Width);
        OnPropertyChanged("MissesLeft");

        _timer.Interval = _model.TimeBetweenBees;
        _timer.Start();

        GameOver = Visibility.Collapsed;
        OnPropertyChanged("GameOver");
    }
```

These properties are bound to the Margin of the flower and hive images, so the ViewModel can move them left and right by updating the left margin and firing a PropertyChanged event.

The ViewModel keeps a reference to an instance of the Model in a private field. Here's where it hooks up its event handlers for the Model's events.

The ViewModel creates each BeeControl and sets the height and width, so it needs a size for that.

The View can start the game by calling the ViewModel's StartGame() method and passing it the sizes of the flower, hive, and play area canvas. The ViewModel sets its private fields, starts the timer, and updates its GameOver property.



As the user swipes, the view's ManipulationDelta event fires, and its event handler calls this method to update the flower's location.

```
    public void ManipulationDelta(double newX) {
        newX = _lastX + newX * 1.5;
        if (newX >= 0 && newX < (_playAreaSize.Width - _flowerSize.Width)) {
            _model.MoveFlower(newX);
            FlowerMargin = new Thickness(newX, 0, 0, 0);
            OnPropertyChanged("FlowerMargin");
            _lastX = newX;
        }
    }

    private void GameOverEventHandler(object sender, EventArgs e) {
        _timer.Stop();
        GameOver = Visibility.Visible;
        OnPropertyChanged("GameOver");
    }

    private void MissedEventHandler(object sender, EventArgs e) {
        OnPropertyChanged("MissesLeft");
    }

    void HiveTimerTick(object sender, EventArgs e) {
        if (_playAreaSize.Width <= 0) return;

        double x = _model.NextHiveLocation();

        HiveMargin = new Thickness(x, 0, 0, 0);
        OnPropertyChanged("HiveMargin");

        BeeControl bee = new BeeControl(x + _hiveSize.Width / 2, 0,
                                        _playAreaSize.Height + _flowerSize.Height / 3, BeeLanded);
        bee.Width = _beeSize.Width;
        bee.Height = _beeSize.Height;
        _beeControls.Add(bee);
    }

    private void BeeLanded(object sender, EventArgs e) {
        BeeControl landedBee = null;
        foreach (BeeControl sprite in _beeControls) {
            if (sprite.FallingStoryboard == sender)
                landedBee = sprite;
        }
        _model.BeeLanded(Canvas.GetLeft(landedBee));
        if (landedBee != null) _beeControls.Remove(landedBee);
    }

    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyChanged(string propertyName) {
        PropertyChangedEventHandler propertyChanged = PropertyChanged;
        if (propertyChanged != null)
            propertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }

    private void PlayerScoredEventHandler(object sender, EventArgs e) {
        OnPropertyChanged("Score");
        _timer.Interval = _model.TimeBetweenBees;
    }
}
```

When the game ends, the ViewModel stops its timer and updates its GameOver property, which is bound to the Visibility of the StackPanel with the Game Over TextBlock, button, and link.

Every time the timer ticks, the ViewModel asks the Model for the next hive location and fires off a new bee by creating a BeeControl and adding it to the _beeControls observable collection, which causes it to get added to the play area Canvas's Children.

A delegate to the `BeeLanded` method is passed into the `BeeControl`, which adds it to the falling animation storyboard's `Completed` event—so when it fires, the `sender` is set to the `Storyboard` object. The `BeeControl`'s `FallingStoryboard` property returns a reference to that `Storyboard` object so the ViewModel can use it to look up the correct `BeeControl` in its `_beeControls` collection.

Whenever the player scores, the timer is updated using the Model's TimeBetweenBees property to make the game go a little faster.

7 CREATE THE BeeAttackGameControl TO MANAGE THE WHOLE GAME.

The View has one more thing in it. `BeeAttackGameControl` has the Image controls for the hive and the flower, the Canvas for the bees to fall down, and the controls that are displayed when the game is over (including a Button to start the game). **Add a new** 

called *BeeAttackGameControl.xaml* to the *View* folder. Here's the XAML:

```
<Grid x:Name="LayoutRoot" Background="SkyBlue">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="10*"/>
        <RowDefinition Height="2*"/>
    </Grid.RowDefinitions>

    <Image x:Name="hive"
           Source="/Assets/Hive (outside).png"
           HorizontalAlignment="Left"
           Margin="{Binding HiveMargin}"/>

    <ItemsControl Grid.Row="1" x:Name="playArea">
        <ItemsControl ItemsSource="{Binding BeeControls}">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <Canvas/>
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
        </ItemsControl>
    </ItemsControl>

    <TextBlock Grid.Row="1" Foreground="Black" VerticalAlignment="Top">
        <Run>Misses left: </Run>
        <Run Text="{Binding MissesLeft}"/>
    </TextBlock>

    <TextBlock Grid.Row="1" Foreground="Black" VerticalAlignment="Top"
               HorizontalAlignment="Right" Text="{Binding Score}"
               Style="{StaticResource PanoramaItemHeaderTextStyle}"/>

    <Image x:Name="flower"
           Source="/Assets/Flower.png"
           Grid.Row="2"
           HorizontalAlignment="Left"
           Margin="{Binding FlowerMargin}"/>

    <StackPanel Grid.Row="1" VerticalAlignment="Center"
                HorizontalAlignment="Center" Visibility="{Binding GameOver}">
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <TextBlock Foreground="Yellow"
                       Style="{StaticResource JumpListAlphabetSmallStyle}">Bee</TextBlock>
            <view:BeeControl Width="75" Height="75"/>
            <TextBlock Foreground="Black"
                       Style="{StaticResource JumpListAlphabetSmallStyle}">Attack</TextBlock>
        </StackPanel>

        <Button Click="Button_Click">Start a new game</Button>

        <HyperlinkButton Content="Learn how to build this game"
                         NavigateUri="http://www.headfirstlabs.com/hfcsharp"
                         TargetName="_blank" />

    </StackPanel>
</Grid>
```

When you create a new Windows Phone user control, the IDE adds it with an empty Grid named LayoutRoot. Change its Background to SkyBlue and give it three rows: the top row for the hive image, the middle row for the play area, and the bottom row for the flower image.

The hive image has its Margin bound to the ViewModel's HiveMargin property.

This ItemsControl's ItemsSource is bound to the observable collection of BeeControl objects in the ViewModel, so each control gets added to the Canvas in the ItemsPanelTemplate.

These TextBlocks show the player's score and the number of misses left before the game ends.

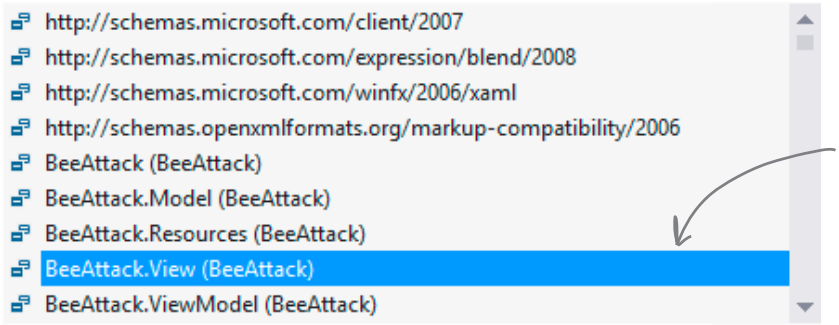
The ViewModel's GameOver property returns a Visibility enum, so it can be bound directly to a XAML Visibility property.

You need the xmlns:view XML namespace markup on the next page for this line. It draws a flapping bee on the page.

The click event handler for this button is in the code-behind on the next page.

Next, add an **`xmlns:view`** property and a **`ManipulationDelta`** event handler to the `<UserControl>` tag at the top of the XAML file. Start by putting your cursor just before the closing **>** bracket on line 10 and hitting Enter to add a line, then **start typing `xmlns:view=""`** —as soon as you hit the quotation mark, an IntelliSense window will pop up to complete the namespace:

```
xmlns:view=""
```



Choose the View namespace from the drop-down. If you chose a different project name, you'll see that project name instead of BeeAttack in the window.

Hit Enter again and start typing **`ManipulationDelta=""`** to add the event handler. The IDE will pop up an IntelliSense window to add a new event handler method to the code-behind:

```
xmlns:view="clr-namespace:BeeAttack.View"
ManipulationDelta=""
```

When you're done, you'll have these two additional lines in the opening `<UserControl>` tag:

```
xmlns:view="clr-namespace:BeeAttack.View"
ManipulationDelta="UserControl_ManipulationDelta"
```

If the IDE doesn't pop up the IntelliSense windows for some reason, you can just type in these lines manually.

Finally, **open *BeeAttackGameControl.xaml.cs* and add the code-behind**. Here's the code for it:

```
using ViewModel;

public partial class BeeAttackGameControl : UserControl {
    private readonly ViewModel.BeeAttackViewModel _viewModel = new ViewModel.BeeAttackViewModel();

    public BeeAttackGameControl() {
        InitializeComponent();

        DataContext = _viewModel;
    }

    private void Button_Click(object sender, RoutedEventArgs e) {
        _viewModel.StartGame(flower.RenderSize, hive.RenderSize, playArea.RenderSize);
    }

    private void UserControl_ManipulationDelta(object sender,
                                               System.Windows.Input.ManipulationDeltaEventArgs e) {
        _viewModel.ManipulationDelta(e.DeltaManipulation.Translation.X);
    }
}
```

The game control has an instance of the ViewModel object, which it uses for its data context.

The Start button calls the ViewModel's StartGame() method, passing the sizes that the ViewModel needs as arguments.

The ManipulationDelta event handler calls straight through to the ViewModel's method.

8 UPDATE THE MAIN PAGE TO ADD THE GAME CONTROL.

Now that we've encapsulated the entire game behind the `BeeAttackGameControl`, we just need to add it to the main page. Open *MainPage.xaml* and **add the `xmlns:view` namespace to the `<phone:PhoneApplicationPage>` opening tag**, just like you did with the opening tag in *BeeAttackGameControl.xaml*.

```
xmlns:view="clr-namespace:BeeAttack.View"
```

Again, if you used a different project name, the namespace will match your project name instead of BeeAttack.

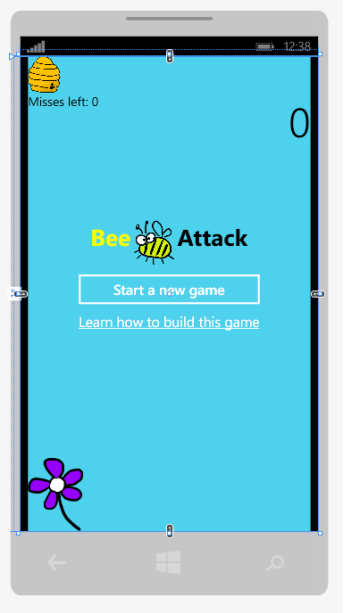
Next, find the Grid named ContentPanel and add your `BeeAttackGameControl` to it. Put your cursor between the opening and closing tags, **start typing `<view:`** to pop up an IntelliSense window, and select `BeeAttackGameControl`:

```
<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <view:
</Grid>
```

BeeAttackGameControl
BeeControl

Here's what the XAML should look like after it's added:

```
<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <view:BeeAttackGameControl/>
</Grid>
```



Congratulations—your game works!

As soon as you update the *MainPage.xaml* file, you should see your game's start page show up in the designer window. Now you can run the game in the emulator or on the device…and you can get creative by expanding the game!

- Try adding bonus bees that are worth more, or evil bees that the player must avoid.
- Make some of the bees fly side to side by adding `(Canvas.Left)` animations.
- We included a "hive interior" image file. Can you think of something cool to do with it?
- **Claim your bragging rights** by publishing your code on CodePlex, GitHub, or another social code sharing site…and then let other readers know about it on the *Head First C#* forum: http://www.headfirstlabs.com/hfcsharp.