**HOW DO I …**

Turn an XNA game into a Silverlight/XNA hybrid application?

In this first of two videos, we’re going to show how you can take an existing XNA game and transform it so that it uses the new Silverlight/XNA application model.

# Overview

We will start with an overview of the existing code, covering both the gameplay and the game’s code.

The game starts up with a main menu that allows us to advance to the main game. We control the tank at the bottom of the screen by tilting the phone left or right, and tap the screen to shoot at the aliens moving across the top of the screen, all the while avoiding the aliens’ own attacks.

The game’s code is pretty straightforward and is composed of several screen classes tied together using the game state management code as seen in the following sample: <http://create.msdn.com/en-US/education/catalog/sample/game_state_management>.

The game is composed of the following screens:

* Loading screen – Responsible for loading the game’s resources and advancing to the main menu screen.
* Background screen – Displays a background for the loading and main menu screen.
* Main menu screen – Allows the player to start playing or to quit the game.
* Gameplay screen – Presents the player with the actual game.

Additionally, the game contains a particle system responsible for displaying various explosions and the dirt kicked by the tank’s tracks and a helper class for managing the game’s state called “GameplayHelper”.

# Upgrading to Mango

Upgrading the project to Windows Phone Mango is as simple as right clicking the project file and selecting the appropriate option from the context menu. Note that this is a one way process and projects cannot be downgraded later on.

The game does not contain any code to support proper tombstoning, but it can still benefit from Mango’s new fast application switching. See, however, how restoring the game brings the player right back to the action, without giving him a chance to prepare himself.

# Silverlight/XNA

In order to properly handle FAS and tombstoning we should add a pause screen so that the user may get ready before returning to the game.

Adding this new screen, with an additional menu, can prove cumbersome while relying on the game state management code for transitioning between various screens. Instead, let us make use of the new application model made available by Windows Phone Mango – Silverlight and XNA hybrid applications.

In Silverlight/XNA applications, it is possible to present both Silverlight and XNA elements on the screen at the same time, as well as transition between multiple pages just like in regular Silverlight applications. Transitioning to this new application model requires some initial effort, but adding screens later on will become a much simpler affair.

1. Start by creating a new hybrid application project in a new solution.
2. See that the project contains two pages at the moment, a main page, which we will turn into a combination of the background and main menu pages, and the game page, which will contain the actual game logic.
3. Note that the newly created solution contains a content project much like an XNA game project does. Add all the assets from the existing version of the game to the new solution. We won’t necessarily be using all these new assets, such as the title font.
4. Update the game’s icon with that from the original game.
5. Start altering MainPage.xaml to function as the game’s main menu. Start by setting the “LayoutRoot” grid’s background to the menu’s background image, “background.png”.
6. Replace the page title with the image containing the game’s name. Replace the stackpanel named “TitlePanel” with the following XAML:

<Image Stretch="None" Margin="40" Source="/Images/title.png" />

1. Change the text inside the single on-screen button to “START GAME”. There is no way to exit a sliverlight application from code, so there is no point in adding a “Quit” menu item.
2. Add the files “ParticleSystem.cs” and “GameplayHelper.cs” from the original game to the new AlienGame project.
3. Add a reference to **Microsoft.Devices.Sensors**.
4. Open **GamePage.xaml.cs** and add the following fields to the **GamePage** class. These are equivalent to the fields from the original games:

// Input Members

SensorReadingEventArgs<AccelerometerReading> accelState;

Accelerometer Accelerometer;

// Create a helper instance for handling game logic

GameplayHelper gameplayHelper;

bool backPressed;

1. Add the following piece of code to the **GamePage**’s constructor, just after the call to “InitializeComponent”. This is nearly identical to the code in the original game that initializes the accelerometer. The code also registers for events that fire when the user presses the device’s back key:

Accelerometer = new Accelerometer();

if (Accelerometer.State == SensorState.Ready)

{

    Accelerometer.CurrentValueChanged += (s, e) =>

    {

        accelState = e;

    };

    Accelerometer.Start();

}

this.BackKeyPress += new EventHandler<System.ComponentModel.CancelEventArgs>(GamePage\_BackKeyPress);

1. Add a handler for the back key presses:

void GamePage\_BackKeyPress(object sender, System.ComponentModel.CancelEventArgs e)

{

    backPressed = true;

    e.Cancel = true;

}

1. Add the following **InitializeGame** method to the **GamePage** class. This method is nearly identical to the **LoadContent** method in the original game:

public void InitializeGame()

{

    gameplayHelper = new GameplayHelper(contentManager, spriteBatch,

        SharedGraphicsDeviceManager.Current.GraphicsDevice);

    gameplayHelper.LoadContent();

gameplayHelper.Start();

}

1. Add some code to the **OnNavigatedTo** method. Place the call exactly where the appropriate placeholder comment is (the one starting with “TODO”):

backPressed = false;

InitializeGame();

1. In **OnNavigatedFrom**, add the following call to the top of the method:

gameplayHelper.UnloadContent();

1. Add the following two methods to the **GamePage** class. They are almost identical to the input handling methods in the original game:

public void HandleInput(bool shouldPause)

{

    Vector3 accelerationInfo = accelState == null ? Vector3.Zero : accelState.SensorReading.Acceleration;

    if (gameplayHelper.HandleInput(shouldPause, accelerationInfo, TouchPanel.GetState()))

    {

        FinishCurrentGame();

    }

}

private void FinishCurrentGame()

{

    backPressed = false;

    NavigationService.GoBack();

}

1. Add the following code to the body of the **GamePage**’s **OnUpdate** method:

HandleInput(backPressed);

float elapsedSeconds = (float)e.ElapsedTime.TotalSeconds;

gameplayHelper.Update(elapsedSeconds);

1. Add the following code to the body of the **GamePage**’s **OnDraw** method:

float elapsedSeconds = (float)e.ElapsedTime.TotalSeconds;

gameplayHelper.Draw(elapsedSeconds);

# Summary

In this first of two videos video, we saw how you can take an XNA game and transform it to a Silverlight/XNA hybrid application. In the next video, we will see how this new object model makes it amazingly simple to solve the game’s remaining issues by providing proper tombstoning and fast application switching while leveraging Silverlight’s page navigations.