# A Summoner’s Tale
# Chapter B
# Going Mobile!

Welcome to my Summoner’s Tale tutorial series on creating a Pokémon-inspired game with MonoGame. I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. The code should work on previous versions of MonoGame and Visual Studio. I plan on creating the editors on macOS and Windows. I’m unfamiliar with Linux, so a few projects may not be done for that platform.

The tutorials will make more sense if they are read in order. You can find the list of tutorials on my web blog, A Summoner’s Tale page. In addition to the PDFs, I will make the code for each tutorial available on GitHub here: https://github.com/Synammon/summoners-tale. It will also be included on the page that links to the tutorials.

In this tutorial, I will be going mobile! Adding projects for Android and iOS. Keep in mind, that iOS is only applicable if you have a Mac computer. Otherwise, you will not be able to build. It will be pretty, but useless.

One of the first pieces that we will need, other than the projects, is a way to handle touch input. This is because the vast majority of devices will not have external devices attached to them. The player will only have their touch screen. Replace the existing Xin class with the following code.

```
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework;
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
using Microsoft.Xna.Framework.Input.Touch;

namespace SummonersTale
{
    public enum MouseButton { Left, Right };

    public class Xin : GameComponent
    {
        private static KeyboardState keyboardState;
        private static KeyboardState lastKeyboardState;
        private static MouseState mouseState;
        private static TouchCollection lastTouchLocations;
        private static MouseState lastMouseState;
        private static TouchCollection touchLocations;

        public static KeyboardState KeyboardState { get { return keyboardState; } }
        public static MouseState MouseState { get { return mouseState; } }

        public static KeyboardState LastKeyboardState { get { return 
lastKeyboardState; } }
        public static MouseState LastMouseState { get { return lastMouseState; } }
        public static Point MouseAsPoint
        {
            get { return new Point(MouseState.X, MouseState.Y); }
        }

        public static Point LastMouseAsPoint
        {
            get { return new Point(LastMouseState.X, LastMouseState.Y); }
        }

        public Xin(Game game) : base(game)
        {
            keyboardState = Keyboard.GetState();
            mouseState = Mouse.GetState();
        }

        public override void Update(GameTime gameTime)
        {
            lastKeyboardState = keyboardState;
            lastMouseState = mouseState;

            keyboardState = Keyboard.GetState();
            mouseState = Mouse.GetState();
            lastTouchLocations = touchLocations;
            touchLocations = TouchPanel.GetState();

            base.Update(gameTime);
        }

        public static bool IsKeyDown(Keys key)
        {
            return keyboardState.IsKeyDown(key);
        }

        public static bool WasKeyDown(Keys key)
        {
            return lastKeyboardState.IsKeyDown(key);
        }

        public static bool WasKeyPressed(Keys key)
        {
            return keyboardState.IsKeyDown(key) && lastKeyboardState.IsKeyUp(key);
        }

        public static bool WasKeyReleased(Keys key)
        {
            return keyboardState.IsKeyUp(key) && lastKeyboardState.IsKeyDown(key);
        }

        public static bool IsMouseDown(MouseButtons button)
        {
            return button switch
            {
                MouseButtons.Left => mouseState.LeftButton == ButtonState.Pressed,
                MouseButtons.Right => mouseState.RightButton == ButtonState.Pressed,
                _ => false,
            };
```

```
        }

        public static bool WasMouseDown(MouseButtons button)
        {
            return button switch
            {
                MouseButtons.Left => lastMouseState.LeftButton ==
ButtonState.Pressed,
                MouseButtons.Right => lastMouseState.RightButton ==
ButtonState.Pressed,
                _ => false,
            };
        }

        public static bool WasMousePressed(MouseButtons button)
        {
            return button switch
            {
                MouseButtons.Left => mouseState.LeftButton == ButtonState.Pressed &&
lastMouseState.LeftButton == ButtonState.Released,
                MouseButtons.Right => mouseState.RightButton == ButtonState.Pressed
&& lastMouseState.RightButton == ButtonState.Released,
                _ => false,
            };
        }

        public static bool WasMouseReleased(MouseButtons button)
        {
            return button switch
            {
                MouseButtons.Left => mouseState.LeftButton == ButtonState.Released
&& lastMouseState.LeftButton == ButtonState.Pressed,
                MouseButtons.Right => mouseState.RightButton == ButtonState.Released
&& lastMouseState.RightButton == ButtonState.Pressed,
                _ => false,
            };
        }

        public static List<Keys> KeysPressed()
        {
            List<Keys> keys = new();

            Keys[] current = keyboardState.GetPressedKeys();
            Keys[] last = lastKeyboardState.GetPressedKeys();

            foreach (Keys key in current)
            {
                if (!last.Contains(key))
                {
                    keys.Add(key);
                }
            }

            return keys;
        }

        public static List<Keys> KeysReleased()
        {
            List<Keys> keys = new();

            Keys[] current = keyboardState.GetPressedKeys();
            Keys[] last = lastKeyboardState.GetPressedKeys();

            foreach (Keys key in current)
            {
                if (last.Contains(key))
                {
                    keys.Add(key);
                }
            }

            return keys;
        }

        public static TouchCollection TouchPanelState
        {
            get { return touchLocations; }
        }

        public static TouchCollection LastTouchPanelState
        {
            get { return lastTouchLocations; }
        }

        public static bool TouchReleased()
        {
            TouchCollection tc = touchLocations;

            if (tc.Count > 0 &&
                tc[0].State == TouchLocationState.Released)
            {
                return true;
            }

            return false;
        }

        public static bool TouchPressed()
        {
            return (touchLocations.Count > 0 &&
                (touchLocations[0].State == TouchLocationState.Pressed));
        }

        public static bool TouchMoved()
        {
            return (touchLocations.Count > 0 &&
                (touchLocations[0].State == TouchLocationState.Moved));
        }

        public static Vector2 TouchLocation
        {
            get
            {
                Vector2 result = Vector2.Zero;

                if (touchLocations.Count > 0)
                {
                    if (touchLocations[0].State == TouchLocationState.Pressed ||
                        touchLocations[0].State == TouchLocationState.Moved)
                    {
                        result = touchLocations[0].Position;
                    }
                }

                return result;
            }
        }

        public static bool WasKeyPressed()
        {
            return
                keyboardState.GetPressedKeyCount() > 0 &&
                lastKeyboardState.GetPressedKeyCount() == 0;
        }
    }
}
```

I added two static fields touchLocations and lastTouchLocations. They hold the current touchpoints and the touchpoints in the last frame. I say touchpoints because, with touch input, there can be multiple points touched and once. I am only concerned about the first touchpoint at this time. There are properties to expose the value of the touchpoint fields.

The first method I added is TouchReleased. It returns if the first touchpoint has been released since the last frame of the game. It works a little differently than you would expect. You don't compare the current frame to the previous frame. What you do is count the number of touchpoints using the Count property of the TouchCollection. If that is greater than zero and the state of the first item is in the Released state, then the first finger to touch the screen has been released.

The TouchPressed method is similar to the Touch released method. It checks to see if the number of touches is greater than zero. If the State property of the first touch point is Pressed, then there had been a new touch.

Sometimes you need to know if a touch has moved. You do that by checking to see if the Count property of the TouchCollection is greater than zero and if the state of the first touch is Moved.

There is then a property for the location of a touch. First, I create a local variable and set it to the zero vector. I check to see if there is a touch. If there is, I check to see if the state of the first touch is Pressed or if it is Moved. If that is true, I set the return value to the Position property of the touch. I finally return either the zero vector or the position of the touch.

I also included a new keyboard method, WasKeyPressed, that checks to see if any key has been pressed. I check that by seeing if the number of keys pressed in the last frame is zero and the number of keys pressed in this frame is greater than zero.

We’re off to a solid start. I am using my PC for development. So, I need to connect to my MacBook in order to build and test iOS. The process for doing that is in Visual Studio to select Tools -> iOS -> Pair to Mac. This may not work if you are on macOS Ventura. There was a change in the way that macOS works with SSH. You need to make an adjustment to a configuration file: /etc/ssh/sshd_config. Add the following two lines at the end and restart your Mac.

```
HostkeyAlgorithms +ssh-rsa
PubkeyAcceptedAlgorithms +ssh-rsa
```

The other piece of the puzzle is that you need to grant Remote Login to your account on your Mac. On your Mac, press CMD-Space and then enter Remote Login. Grant permission to a group your account belongs to or your actual account.

From here, go back to Visual Studio on your PC. Go to Tools -> iOS -> Pair to Mac and follow the instructions. There is one last piece I did. I found that the simulator for iOS would often crash and not be able to start back up again on my PC. So, what I ended up doing was setting things so that the simulator would start up on my Mac. To do that, go to Tools -> Options -> Xamarin -> iOS Settings and deselect Remote Simulator to Windows.

Let’s add the iOS project. Right-click the SummonersTale in the Solution Explorer, select Add and then New Project. Enter MonoGame in the search box. Select MonoGame iOS Application and click Next. Name this new project SummonersTaleGameiOS and click create. Once the project is added, there are a few configuration steps that we need to follow. First, we need to make it the startup project. Right-click the SummonersTaleGameiOS project and select Set As Startup Project. Next, there is a discrepancy between the project and the Info.plist. The project is set for version 11.2, and the Info.plist is set for 11.0. Open the info.plist by double-clicking it. Change the 11.0 to 11.2. The last thing to do is reference our two libraries. Right-click the SummonersTaleGameiOS project and select Add Reference. Select the Psilibrary project. Before closing it, expand the Shared node and select SummonersTale. There is a problem, however. Game.Exit is not used in iOS. We are using it in two places in the SummonersTale shared project. The first is the LeaveButton_Click method, and the other is in the Update method. Comment out both lines. If you build and run now, you will be presented with the regular cornflower blue screen that we all love!

Let’s move on to Android. It is really the easier of the two options, because a second computer isn’t required. You do, however, need a physical or virtual device for debugging. For a virtual device, it is recommended that you have Hyper-V enabled and virtualization enabled in the BIOS. For Hyper-V, you will need to be running Windows Professional. If you can’t fulfil those criteria, all is not lost! You can connect a physical device through a USB data cable.

I will walkthrough creating a virtual device. So, how do you create a virtual device? That is done by selecting Tools -> Android -> Android Device Manager. You may be prompted to create a default device if you haven’t created a device before. This is perfectly fine. I created a second device. From the window that comes up, click the New button. In the list that comes up, click the drop-down for Base Device and select Pixel 3 XL, the 3 XL will always have a sweet spot in my heart. Then click the Create button to create the device. It should be relatively quick. Close the Android Virtual Device window.

From now on, when talking about Android, I will say device instead of a physical device or virtual device. Just a little less confusion and typing. Make sure that your device is powered on and connected to the computer.

We are going to follow essentially the same steps that we took in adding an iOS project. Right-click SummonersTale in the solution explorer, select Add and then New Project. Enter MonoGame in the search box. Select MonoGame Android Application and click Next. Name this new project SummonersTaleGameAndroid and click Create.

Once the project is created, we need to reference the projects, as we did before. Right-click the SummonersTaleGameAndriod project and select Add Reference. Select the Psilibrary project to reference it. Before you close the window, expanded the Shared node and select the SummonersTale project to reference that as well. Finally, right-click the SummonersTaleGameAndroid project and select Set As Startup Project. Now, run the solution. Everything will compile without error or warning and you will be presented with the familiar cornflower blue screen.

Things are getting confusing. We have three Game1 classes in three different projects. I resolved this by renaming them. You can right-click the class name in the solution explorer and select Rename. When prompted, choose to rename the instances to the new name. For SummonersTaleGameAndroid, rename it to Android. For SummonersTaleGameiOS, rename it to iOS. Finally, for the last project, rename it to Desktop.

Before continuing, there are a few changes I want to make. First, I want to update the Button class to use touch input. Replace the Button class with the following code.

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input.Touch;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework;
using SummonersTale.Forms;
using System;
using System.Collections.Generic;
using System.Text;
using static System.Net.Mime.MediaTypeNames;

namespace SummonersTale
{
    public enum ButtonRole { Accept, Cancel, Menu }

    public class Button : Control
    {
        #region
        public event EventHandler Click;

        #endregion
        #region Field Region

        private readonly Texture2D _background;
        float _frames;

        public ButtonRole Role { get; set; }
        public int Width { get { return _background.Width; } }
        public int Height { get { return _background.Height; } }

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public Button(Texture2D background, ButtonRole role)
        {
            Role = role;
            _background = background;
        }

        #endregion

        #region Method Region

        public override void Draw(SpriteBatch spriteBatch)
        {
            Rectangle destination = new(
            (int)Position.X,
                (int)Position.Y,
                _background.Width,
                _background.Height);

            spriteBatch.Draw(_background, destination, Color.White);

            _spriteFont = ControlManager.SpriteFont;

            Vector2 size = _spriteFont.MeasureString(Text);
            Vector2 offset = new((_background.Width - size.X) / 2,
((_background.Height - size.Y) / 2));

            spriteBatch.DrawString(_spriteFont, Text, Helper.NearestInt((Position +
offset)), Color);
        }

        public override void HandleInput()
        {
            MouseState mouse = Mouse.GetState();
            Point position = new(mouse.X, mouse.Y);
            Rectangle destination = new Rectangle(
                (int)(Position.X + Offset.X),
                (int)(Position.Y + Offset.Y),
                _background.Width,
                _background.Height);
            if ((Role == ButtonRole.Accept && Xin.WasKeyReleased(Keys.Enter)) ||
                (Role == ButtonRole.Accept && Xin.WasKeyReleased(Keys.Space)))
            {
                OnClick();
                return;
            }

            if (Role == ButtonRole.Cancel && Xin.WasKeyReleased(Keys.Escape))
            {
                OnClick();
                return;
            }

            if (Xin.WasMouseReleased(MouseButton.Left) && _frames >= 5)
            {
                Rectangle r = destination.Scale(Settings.Scale);

                if (r.Contains(position))
                    OnClick();
            }

            if (Xin.TouchReleased() && _frames >= 5)
            {
                Rectangle rectangle= destination.Scale(Settings.Scale);

                if (rectangle.Contains(Xin.TouchLocation))
                {
                    OnClick();
                }
            }
        }

        private void OnClick()
        {
            Click?.Invoke(this, null);
        }

        public override void Update(GameTime gameTime)
        {
            _frames++;
            HandleInput();
        }

        public void Show()
        {
            _frames = 0;
        }

        #endregion
    }
}
```

What the new code does is check if the touch screen has been released and if at least five frames have passed since the last input event. If that is true, it creates a scaled rectangle like it did when checking for a mouse click. Then, if the touch location is inside the rectangle, it calls the OnClick method to trigger the click event if it is subscribed to.

The next change was to the RightLeftSelector. What I did was update the HandleMouseInput method to handle touch events as well as mouse clicks. Similarly to buttons, I check to see if there has been a touch release event. If there has, I check to see if the buttons contain the touch location and proceed the same as if they contained the mouse. Replace the HandleMouseInput method of the RightLeftSelector with the following code.

```
private void HandleMouseInput()
{
    if (Xin.WasMouseReleased(MouseButton.Left))
    {
        Point mouse = Xin.MouseAsPoint;

        if (_leftSide.Scale(Settings.Scale).Contains(mouse))
        {
            _selectedItem--;
            if (_selectedItem < 0)
                _selectedItem = this.Items.Count - 1;
            OnSelectionChanged();
        }

        if (_rightSide.Scale(Settings.Scale).Contains(mouse))
        {
            _selectedItem++;
            if (_selectedItem >= _items.Count)
                _selectedItem = 0;
            OnSelectionChanged();
        }
    }

    if (Xin.TouchReleased())
    {
        if (_leftSide.Scale(Settings.Scale).Contains(Xin.TouchLocation))
        {
            _selectedItem--;
            if (_selectedItem < 0)
                _selectedItem = this.Items.Count - 1;
            OnSelectionChanged();
        }

        if (_rightSide.Scale(Settings.Scale).Contains(Xin.TouchLocation))
        {
            _selectedItem++;
            if (_selectedItem >= _items.Count)
                _selectedItem = 0;
            OnSelectionChanged();
        }
    }
}
```

Text input is tricky. In fact, I've not found an easy way to inject the onscreen keyboard into my text box. Then again, I haven't put much effort into it. I could create a virtual keyboard, and that might be the method that I opt for in order to solve that problem. What I've done in the past is bypass keyboard input entirely, either by enforcing a static name or choosing randomly from a list.

So, there is a problem with iOS at the moment. When building, content is not copied over. You will get an exception, "Content item could not be found." Fortunately, there is an easy solution. You need to update the build targets to copy the files. The file is located here:

"C:\Users\{username}\.nuget\packages\monogame.content.builder.task\3.8.1.303\build\MonoGame.Content.Builder.Task.targets"

You just need to add a line to it. Replace that file with the following.

```
<Project>

  <!-- Add MonoGameContentReference to item type selection in Visual Studio -->
  <ItemGroup>
    <AvailableItemName Include="MonoGameContentReference" />
  </ItemGroup>

  <!-- This disables the IDE feature that skips executing msbuild in some build
situations. -->
  <PropertyGroup>
    <DisableFastUpToDateCheck>true</DisableFastUpToDateCheck>
       <CollectBundleResourcesDependsOn> IncludeContent;
</CollectBundleResourcesDependsOn>
  </PropertyGroup>

  <!--
    Target flow
      1. CollectContentReferences
      2. PrepareContentBuilder
      3. RunContentBuilder
      4. IncludeContent
  -->

  <!--
    ========================
    CollectContentReferences
    ========================

    Converts MonoGameContentReference items to ContentReference items, deriving the
necessary metadata.

    Outputs:
      - ContentReference: references to .mgcb files that can be built with MGCB
        - FullDir: the absolute path of the folder containing the .mgcb file
        - ContentDir: the relative path of the resource folder to contain the content
files
        - ContentOutputDir: the absolute path of the bin folder containing final built
content
        - ContentIntermediateOutputDir: the absolute path of the obj folder containing
intermediate content
```

```
    Example:
      - Given the following file setup:
        - C:\Game\Game.Shared\Content.mgcb
        - C:\Game\Game.DesktopGL\Game.DesktopGL.csproj
          - MonoGameContentReference: ..\Game.Shared\Content.mgcb
      - Output:
        - ContentReference
          - FullDir: C:/Game/Game.Shared/
          - ContentDir: Game.Shared/
          - ContentOutputDir: C:/Game/Game.Shared/bin/DesktopGL/Content
          - ContentIntermediateOutputDir: C:/Game/Game.Shared/obj/DesktopGL/Content
  -->
  <Target Name="CollectContentReferences">

    <ItemGroup Condition="'$(EnableMGCBItems)' == 'true'">
      <MonoGameContentReference Include="**/*.mgcb" />
    </ItemGroup>

    <ItemGroup>

      <!-- Start with existing metadata. -->
      <ContentReference Include="@(MonoGameContentReference)">
        <Link>%(MonoGameContentReference.Link)</Link>

<FullDir>%(MonoGameContentReference.RootDir)%(MonoGameContentReference.Directory)</FullDi
r>
        <ContentFolder>%(MonoGameContentReference.ContentFolder)</ContentFolder>
        <OutputFolder>%(MonoGameContentReference.Filename)</OutputFolder>
      </ContentReference>

      <!--
        Process intermediate metadata.
        Switch all back-slashes to forward-slashes so the MGCB command doesn't think it's
trying to escape characters or quotes.
        ContentFolder will be the name of the containing folder (using the Link if it
exists) so the directory structure of the included content mimics that of the source
content.
      -->
      <ContentReference>
        <FullDir>$([System.String]::Copy("%(FullDir)").Replace('\','/'))</FullDir>
        <ContentFolder Condition="'%(ContentFolder)' == '' AND '%(Link)' !=
''">$([System.IO.Path]::GetDirectoryName(%(Link)))</ContentFolder>
        <ContentFolder Condition="'%(ContentFolder)' == '' AND '%(Link)' == '' AND
'%(RelativeDir)' !=
''">$([System.IO.Path]::GetFileName($([System.IO.Path]::GetDirectoryName(%(RelativeDir)))
))</ContentFolder>
      </ContentReference>

      <!-- Assemble final metadata. -->
      <ContentReference>
        <ContentDir>%(ContentFolder)/</ContentDir>

<ContentOutputDir>%(FullDir)bin/$(MonoGamePlatform)/%(OutputFolder)</ContentOutputDir>

<ContentIntermediateOutputDir>%(FullDir)obj/$(MonoGamePlatform)/$(TargetFramework)/%(Outp
utFolder)</ContentIntermediateOutputDir>
      </ContentReference>
    </ItemGroup>

  </Target>

  <!--
    =====================
    PrepareContentBuilder
    =====================

    Set and validate properties, and create folders for content output.

    Outputs:
      - PlatformResourcePrefix: the platform-specific prefix for included content paths
      - MonoGameMGCBAdditionalArguments: extra arguments to add to the MGCB call
  -->
  <Target Name="PrepareContentBuilder" DependsOnTargets="CollectContentReferences">

    <PropertyGroup>
      <PlatformResourcePrefix Condition="'$(MonoGamePlatform)' ==
'MacOSX'">$(MonoMacResourcePrefix)</PlatformResourcePrefix>
      <PlatformResourcePrefix Condition="'$(MonoGamePlatform)' ==
'iOS'">$(IPhoneResourcePrefix)</PlatformResourcePrefix>
      <PlatformResourcePrefix Condition="'$(MonoGamePlatform)' ==
'Android'">$(MonoAndroidAssetsPrefix)</PlatformResourcePrefix>
      <PlatformResourcePrefix Condition="'$(PlatformResourcePrefix)' != '' And
!HasTrailingSlash('$(PlatformResourcePrefix)')">$(PlatformResourcePrefix)\</PlatformResou
rcePrefix>
      <PlatformResourcePrefix Condition="'$(PlatformResourcePrefix)' ==
''"></PlatformResourcePrefix>
      <MonoGameMGCBAdditionalArguments Condition="'$(MonoGameMGCBAdditionalArguments)' ==
''">/quiet</MonoGameMGCBAdditionalArguments>
    </PropertyGroup>

    <Error
      Text="The MonoGamePlatform property was not defined in the project!"
      Condition="'$(MonoGamePlatform)' == ''" />

    <Warning
      Text="No Content References Found. Please make sure your .mgcb file has a build
action of MonoGameContentReference"
      Condition="'%(ContentReference.FullPath)' == ''" />

    <Warning
      Text="Content Reference output directory contains '..' which may cause content to
be placed outside of the output directory. Please set ContentFolder on your
MonoGameContentReference '%(ContentReference.Filename)' to enforce the correct content
output location."

Condition="$([System.String]::Copy('%(ContentReference.ContentDir)').Contains('..'))" />

    <MakeDir Directories="%(ContentReference.ContentOutputDir)"/>
    <MakeDir Directories="%(ContentReference.ContentIntermediateOutputDir)"/>

  </Target>

  <!--
    =================
    RunContentBuilder
    =================

    Run MGCB to build content and include it as ExtraContent.

    Outputs:
      - ExtraContent: built content files
        - ContentDir: the relative path of the embedded folder to contain the content
files
  -->
  <Target Name="RunContentBuilder" DependsOnTargets="PrepareContentBuilder">

    <!-- Remove this line if they make dotnet tool restore part of dotnet restore build -
->
    <!-- https://github.com/dotnet/sdk/issues/4241 -->
    <Exec Command="$(DotnetCommand) tool restore" />

    <!-- Execute MGCB from the project directory so we use the correct manifest. -->
    <Exec
      Condition="'%(ContentReference.FullPath)' != ''"
      Command="$(DotnetCommand) $(MGCBCommand) $(MonoGameMGCBAdditionalArguments)
/@:&quot;%(ContentReference.FullPath)&quot; /platform:$(MonoGamePlatform)
/outputDir:&quot;%(ContentReference.ContentOutputDir)&quot;
/intermediateDir:&quot;%(ContentReference.ContentIntermediateOutputDir)&quot;
/workingDir:&quot;%(ContentReference.FullDir)&quot;"
      WorkingDirectory="$(MSBuildProjectDirectory)" />

    <ItemGroup>
      <ExtraContent
        Condition="'%(ContentReference.ContentOutputDir)' != ''"
        Include="%(ContentReference.ContentOutputDir)\**\*.*">
        <ContentDir>%(ContentReference.ContentDir)</ContentDir>
      </ExtraContent>
    </ItemGroup>

  </Target>

  <!--
    ==============
    IncludeContent
    ==============

    Include ExtraContent as platform-specific content in the project output.

    Outputs:
      - AndroidAsset: built content files if Android
      - BundleResource: built content files if MacOSX or iOS
      - Content: built content files for all other platforms
  -->
  <Target
    Name="IncludeContent"
    DependsOnTargets="RunContentBuilder"
    Condition="'$(EnableMGCBItems)' == 'true' OR '@(MonoGameContentReference)' != ''"

Outputs="%(ExtraContent.RecursiveDir)%(ExtraContent.Filename)%(ExtraContent.Extension)"
    BeforeTargets="BeforeBuild">

    <!-- Call CreateItem on each piece of ExtraContent so it's easy to switch the item
type by platform. -->
```

```
    <CreateItem
      Include="%(ExtraContent.FullPath)"

AdditionalMetadata="Link=$(PlatformResourcePrefix)%(ExtraContent.ContentDir)%(ExtraConten
t.RecursiveDir)%(ExtraContent.Filename)%(ExtraContent.Extension);CopyToOutputDirectory=Pr
eserveNewest"
      Condition="'%(ExtraContent.Filename)' != ''">
      <Output TaskParameter="Include" ItemName="Content" Condition="'$(MonoGamePlatform)'
!= 'Android' And '$(MonoGamePlatform)' != 'iOS' And '$(MonoGamePlatform)' != 'MacOSX'" />
      <Output TaskParameter="Include" ItemName="BundleResource"
Condition="'$(MonoGamePlatform)' == 'MacOSX' Or '$(MonoGamePlatform)' == 'iOS'" />
      <Output TaskParameter="Include" ItemName="AndroidAsset"
Condition="'$(MonoGamePlatform)' == 'Android'" />
    </CreateItem>

  </Target>

</Project>
```

Okay, now we need to update the Android and iOS classes to run the game. Let's start with the Android one. Make sure it is the startup project. Once it is, update the code to the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SummonersTale;
using SummonersTale.StateManagement;

namespace SummonersTaleGameAndroid
{
    public class Android : Game
    {
        private readonly GameStateManager _manager;
        private readonly GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;

        private GamePlayState _playState;
        private TitleState _titleState;
        private MainMenuState _mainMenuState;
        private NewGameState _newGameState;

        public SpriteBatch SpriteBatch => _spriteBatch;

        public TitleState TitleState => _titleState;
        public GamePlayState PlayState => _playState;

        public Android()
        {
            SummonersTale.Settings.Load();

            _graphics = new GraphicsDeviceManager(this);
            _manager = new GameStateManager(this);

            Services.AddService(typeof(GraphicsDeviceManager), _graphics);

            Content.RootDirectory = "Content";
            IsMouseVisible = true;

            Components.Add(_manager);
        }

        protected override void Initialize()
        {
            SummonersTale.Settings.Resolution = new(
                GraphicsDevice.DisplayMode.Width,
                GraphicsDevice.DisplayMode.Height);

            Components.Add(new Xin(this));

            _graphics.ApplyChanges();

            base.Initialize();
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);
            Services.AddService(typeof(SpriteBatch), _spriteBatch);

            _playState = new(this);
            _titleState = new(this);
            _mainMenuState = new(this);
            _newGameState = new(this);

            _manager.PushState(_titleState);
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            base.Draw(gameTime);
        }
    }
}
```

So, what is new here? Well, instead of setting the back buffer height and width, I get the height and width from the graphics device. In the initialize method, I set the settings to be those values. Why? Well, Android supports only one resolution. You can't set it, no matter how hard you try. For that reason, I grab the values from the graphics device. Fortunately, everything else will just work. At the moment, we don't have to move anything to the project because it all just works, thanks to the shared project. Also, instead of adding the FramesPerSecond instance, I skip it entirely.

If you build and run, things mostly work. The first problem, the time message is not displayed correctly. That is because the area the scene is being rendered to is hardcoded and not using the settings, so they do not match the resolution of the phone. The resolution to that is next. The second is, well, even though you can get to the gameplay screen, everything is painful to do. It stutters, and the sprite is immensely slow. I will be solving that problem in the following tutorial.

Let's fix the problem with the hard-coded value. Replace the Draw method of the TitleScreen with the following code.

```
public override void Draw(GameTime gameTime)
{
    string message = "Game with begin in " + ((int)_timer).ToString() + " seconds.";
    Vector2 size = _spriteFont.MeasureString(message);

    GraphicsDevice.SetRenderTarget(renderTarget);
    GraphicsDevice.Clear(Color.Black);

    Vector2 location = new((TargetWidth - size.X) / 2, TargetHeight -
(_spriteFont.LineSpacing * 5));
    SpriteBatch.Begin();

    SpriteBatch.DrawString(
        _spriteFont,
        message,
        Helper.NearestInt(location),
        Color.White);

    SpriteBatch.End();

    GraphicsDevice.SetRenderTarget(null);

    SpriteBatch.Begin();

    SpriteBatch.Draw(renderTarget, new Rectangle(new(0,0), Settings.Resolution),
Color.White);

    SpriteBatch.End();
    base.Draw(gameTime);
}
```

All it does is pass in the resolution from the settings class instead of new(1920, 1080). All good! Let's switch to iOS if you're following along with that platform. First, right-click the SummonersTaleGameiOS project and select Set as Startup Project. Now, replace the iOS class with the following version.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SummonersTale;
using SummonersTale.StateManagement;

namespace SummonersTaleGameiOS
{
    public class iOS : Game
    {
        private readonly GameStateManager _manager;
        private readonly GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;

        private GamePlayState _playState;
        private TitleState _titleState;
        private MainMenuState _mainMenuState;
        private NewGameState _newGameState;

        public SpriteBatch SpriteBatch => _spriteBatch;

        public TitleState TitleState => _titleState;
        public GamePlayState PlayState => _playState;

        public iOS()
        {
            SummonersTale.Settings.Load();

            _graphics = new GraphicsDeviceManager(this);
            _manager = new GameStateManager(this);

            Services.AddService(typeof(GraphicsDeviceManager), _graphics);

            Content.RootDirectory = "Content";
            IsMouseVisible = true;

            Components.Add(_manager);
        }

        protected override void Initialize()
        {
            SummonersTale.Settings.Resolution = new(
                GraphicsDevice.DisplayMode.Width,
                GraphicsDevice.DisplayMode.Height);

            Components.Add(new Xin(this));

            _graphics.ApplyChanges();

            base.Initialize();
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);
            Services.AddService(typeof(SpriteBatch), _spriteBatch);

            _playState = new(this);
            _titleState = new(this);
            _mainMenuState = new(this);
            _newGameState = new(this);
            _manager.PushState(_titleState);
        }

        protected override void Update(GameTime gameTime)
        {
            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            base.Draw(gameTime);
        }
    }
}
```

It is nearly identical to the Android class. In fact, the only differences are the namespace and the fact that I removed the call to Exit that Visual Studio was complaining about. If you build and run, things will work mostly as expected. The one issue is that you can't see the sprite in the upper left-hand corner. Also, you can't move at all. Again, that will be addressed in the following tutorial.

So, we've gone mobile. Things are working basically the same on all platforms. I'm not going to dive any further into this tutorial. I think I've fed you more than enough for one day. So, that is going to be it for this tutorial. I covered a lot, and I don't want to overload you. I encourage you to keep visiting my blog for the latest news on my tutorials.

Good luck with your game programming adventures.
*Cynthia*