Eyes of the Dragon Tutorials 4.0

Part 3

Enter Android!

I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the Eyes of the Dragon 4.0 page of my web blog. I will be making the project available on GitHub here. It will be included on the page that links to the tutorials.

This is part three in a series of tutorials I plan to write on creating a role-playing game with MonoGame. I've worked on similar tutorials in the past using XNA and the past version of MonoGame 3.8. I have discovered better ways of doing some things in the process of writing more tutorials and had to go back to fix things. I'm hoping in this series not to make those same mistakes. Also, I am going to make the game cross-platform. In my previous tutorials, they were focused on Windows only. I want to open things up for macOS, Linux, Andriod, iOS, and potentially any platform the MonoGame supports.

The main focus of this tutorial will be adding an Android project to the solution and being able to run it on Android. You have two options as far as Android is concerned. You can run the game on an Android emulator, or you can run it on a physical device. If you want to run it on an emulator you will need the professional version of Windows with Hyper-V enabled. You can run it on an emulator without that. However, performance is painful, even on advanced hardware. Personally, I will be using an emulator. The directions for a physical device will be similar.

We need to do a few things to pave the way for Android. We have the makings of supporting multiple resolutions but we aren't quite there yet. We need to talk about how to handle that. The answer is render targets. Much like with a back buffer, rendering it done to an image in a base resolution, 1280 by 720 in our case. This image will then be flipped to the screen and scaled to match the resolution of the target. This is important because Android supports only a single resolution, and the resolution can vary wildly between devices.

Let's get started! First, we need to add an Android project to our solution. Right-click the EyesOfThe-Dragon solution, select Add and then New Project. In the New Project window search for MonoGame. Navigate to the Android option and click on Next. Name this new project EyesOfTheDragonAndroid. Now, we need to reference the libraries. Right-click the EyesOfTheDragonAndriod project, select Add Shared Project Reference. Select the SharedProject project and add it. Right-click the EyesOfTheDrag-onAndroid project and select Add Project Reference. Select the RpgLibrary project and add it. Finally, right-click the EyesOfTheDragonAndoid project and select Set as Startup Project.

Before we can run the project, we need to build an emulator. If you are using a physical device, you can skip this paragraph. You have opted for the virtual device option. So, you will need to create a vir-tual 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.

Saying Game1 class all the time is going to be confusing. What I will do is rename the first project to Desktop and the Android project to Android. In the EyesOfTheDragon project, select the Game1.cs file and press the F2 key. Change the name to Desktop. When prompted, select Yes to rename code elements. Now, select the Game1 class in the EyesOfTheDragonAndroid project and press F2. Enter Android for the name and rename the code elements. If you aren't prompted to change the code elements, open the Desktop file. Scroll down to Game1 and press Control R, R. Enter Desktop for the new name. Repeat the process for the Android file if necessary.

So far, so good. We have an Android machine, virtual or physical, up and running. If you build and run now, you will be presented with the cornflower blue screen. Let's update the Android project to match the Desktop project. Replace the code for the Android class to the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SharedProject;
using SharedProject.GameScreens;

namespace EyesOfTheDragonAndroid
{
    public class Android : Game
    {
        private GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        public GameStateManager GameStateManager { get; private set; }
        public ITitleState TitleState { get; private set; }
        public IStartMenuState StartMenuState { get; private set; }

        public Android()
        {
            _graphics = new GraphicsDeviceManager(this);

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

            _graphics.PreferredBackBufferWidth = Settings.BaseWidth;
            _graphics.PreferredBackBufferHeight = Settings.BaseHeight;
            _graphics.ApplyChanges();

            Components.Add(new Xin(this));

            GameStateManager = new GameStateManager(this);
            Components.Add(GameStateManager);
            Services.AddService(typeof(GameStateManager), GameStateManager);

            TitleState = new TitleState(this);
            StartMenuState = new StartMenuState(this);
        }
```

```
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);

            Services.AddService(typeof(SpriteBatch), _spriteBatch);
            GameStateManager.PushState((TitleState)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);

            // TODO: Add your drawing code here

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

This is an exact copy and paste from the Desktop, with a change of name for the Android project. Wait a minute, though. What is with all of the squiggly lines? That is because the project is set with a minimum target of Android 23, and most of our code is targeting Android 31. The most straightforward resolution, and the one I am taking, is to upgrade everything to Android 31. To start, we want to change the project properties. Right-click the EyesOfTheDragonAndroid project and select Properties. In the tab that comes up, you will see Supported OS Version. Change that value to 31. Also, I made a change in the AndroidManifest.xml file. Open that file and replace the 23 with 31.

With our emulator or physical device up and running, we can run the game. Do that as you usually would by pressing F5. In the future, when talking about emulators or physical devices, I'm simply going to say device. When we do run the game, though, there are two problems. First, the device is in the wrong orientation. We want landscape. The other is that it is not quite full screen. Both of those are tackled in Activity1 class. Replace the code in that class with the following.

```
using Android.App;
using Android.Content.PM;
using Android.OS;
```

```
using Android.Views;
using Microsoft.Xna.Framework;

namespace EyesOfTheDragonAndroid
{
    [Activity(
        Label = "@string/app_name",
        MainLauncher = true,
        Icon = "@drawable/icon",
        AlwaysRetainTaskState = true,
        LaunchMode = LaunchMode.SingleInstance,
        ScreenOrientation = ScreenOrientation.Landscape,
        ConfigurationChanges = ConfigChanges.Orientation | ConfigChanges.Keyboard | Con-
figChanges.KeyboardHidden | ConfigChanges.ScreenSize
    )]
    public class Activity1 : AndroidGameActivity
    {
        private Android _game;
        private View _view;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            _game = new Android();
            _view = _game.Services.GetService(typeof(View)) as View;
            SetContentView(_view);
            HideSystemUI();
            _game.Run();
        }

        private void HideSystemUI()
        {
            // Apparently for Android OS Kitkat and higher, you can set a full screen
mode. Why this isn't on by default, or some kind
            // of simple switch, is beyond me.
            // Got this from the following forum post: http://community.mo-
nogame.net/t/blocking-the-menu-bar-from-appearing/1021/2
            if (Build.VERSION.SdkInt >= BuildVersionCodes.Kitkat)
            {
                View decorView = Window.DecorView;
                var uiOptions = (int)decorView.SystemUiVisibility;
                var newUiOptions = (int)uiOptions;

                newUiOptions |= (int)SystemUiFlags.LowProfile;
                newUiOptions |= (int)SystemUiFlags.Fullscreen;
                newUiOptions |= (int)SystemUiFlags.HideNavigation;
                newUiOptions |= (int)SystemUiFlags.ImmersiveSticky;

                decorView.SystemUiVisibility = (StatusBarVisibility)newUiOptions;

                this.Immersive = true;
            }
        }
    }
}
```

The first change is I set the screen orientation to Landscape mode. The other addition is that I call a new method that I found on the MonoGame forum for setting an app to full screen. As you can see, it

applies to Kit Kat and above, which is quite old. If we are Kit Kat or above, we proceed. The rest of the method is just setting some style options. If you're interested, I suggest navigating the Android forums for more details.

The image still isn't filling the screen. The target resolution of a Pixel 3 XL is 1440 by 2960. How are we going to get our image to fit? Well, as I said earlier, we will use render targets. To use them, we need to set the screen size from the game in the library. That is where the Settings class comes in, and we have different projects for desktop and mobile.

Let's start by sending the screen size to the Settings class. In the Android and Desktop classes, update the Initialize method to the following.

```
protected override void Initialize()
{
    // TODO: Add your initialization logic here

    Settings.TargetHeight = _graphics.PreferredBackBufferHeight;
    Settings.TargetWidth = _graphics.PreferredBackBufferWidth;

    base.Initialize();
}
```

Also, setting the back buffer height and width has no effect on the graphics device in Android. We need to remove the code for setting them in the constructor. Change the constructor class of the Android class to the following.

```
public Android()
{
    _graphics = new GraphicsDeviceManager(this);

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

    Components.Add(new Xin(this));

    GameStateManager = new GameStateManager(this);
    Components.Add(GameStateManager);
    Services.AddService(typeof(GameStateManager), GameStateManager);

    TitleState = new TitleState(this);
    StartMenuState = new StartMenuState(this);
}
```

There, now the Settings class knows the height and width of the graphics device we are rendering to. However, it is in the format of height and width. It would be useful to have a rectangle object like we do for the base resolution. Update the Settings class to the following.

The next step is to move to use render targets. This is pretty straightforward. You define what your base resolution is. You create a render target in code at that resolution. You switch from rendering to the screen to the render target. You draw your scene to the render target. You switch rendering back

to the screen. Finally, you draw the scene to the screen scaling it to the target resolution. Any game logic and such is done in the base resolution. Let's start with the title state. Update the code of that class to the following.

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using SharedProject.Controls;

namespace SharedProject.GameScreens
{
    public interface ITitleState
    {
        GameState Tag { get; }
    }

    public class TitleState : GameState, ITitleState
    {
        #region Field region

        Texture2D backgroundImage;
        RenderTarget2D renderTarget;

        readonly GameState tag;

        public GameState Tag
        {
            get { return tag; }
        }

        #endregion

        #region Constructor region

        public TitleState(Game game)
            : base(game)
        {
            Game.Services.AddService<ITitleState>(this);
            SpriteBatch = Game.Services.GetService<SpriteBatch>();
        }

        #endregion

        #region XNA Method region

        protected override void LoadContent()
        {
            base.LoadContent();

            renderTarget = new(GraphicsDevice, Settings.BaseWidth, Settings.BaseHeight);

            ContentManager Content = Game.Content;
            backgroundImage = Content.Load<Texture2D>(@"Backgrounds\titlescreen");

            LinkLabel startLabel = new();
            startLabel.Position = new Vector2(350, 600);
            startLabel.Text = "Press ENTER to begin";
            startLabel.Color = Color.White;
```

```
            startLabel.TabStop = true;
            startLabel.HasFocus = true;
            startLabel.Selected += StartLabel_Selected; ;

            ControlManager.Add(startLabel);
        }

        private void StartLabel_Selected(object sender, EventArgs e)
        {
            StartMenuState state = (StartMenuState)Game.Services.GetService<IStartMen-
uState>().Tag;
            StateManager.ChangeState(state);
        }

        public override void Update(GameTime gameTime)
        {
            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            GraphicsDevice.SetRenderTarget(renderTarget);
            renderTarget.GraphicsDevice.Clear(Color.Black);

            SpriteBatch.Begin();

            SpriteBatch.Draw(
                backgroundImage,
                Settings.BaseRectangle,
                Color.White);

            base.Draw(gameTime);

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin();

            SpriteBatch.Draw(renderTarget, Settings.TargetRectangle, Color.White);

            SpriteBatch.End();
        }

        protected override void Show()
        {
            base.Show();

            LoadContent();
        }

        #endregion
    }
}
```

What I did was add a new RenderTarget2D field. In the LoadContent method, I initialize it with the current graphics device and the base height and width. Then, in the Draw method I follow the process that I described earlier. I set the render target to the new render target field and fill it with black. I then render the scene. After rendering the scene, I set the render target back to the screen by passing

in null. I then call Begin to start rendering. I draw the render target at the target resolution. And that it is. We are now going to do everything as if the screen was in 1280 by 720 mode and then render that to the window of our device. If there is anything that is platform specific, we will handle that in that project. Trying to share as much code as possible between our solutions.

Well, it is all fine and good to share code between the platforms. Sometimes, you need to override the behaviour in a platform. One such instance is Link Labels. Typically, Android devices don't have keyboards attached to them. So, our game is stuck on the title screen. Enter the need to override the title screen in the Android and iOS games. I won't be introducing iOS at this time. That will be a future tutorial. So, what I will do is introduce simple touch input. That is single taps on the screen. In the future, I will introduce gestures.

Because all devices could potentially have touch input, I will add touch input to Xin. Replace the code in the Xin class in the SharedProject project.

```
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 SharedProject
{
    public enum MouseButtons { 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 lastKeyboard-
State; } }
        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 had 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.

So, I'm going to update the title screen. I'm going to make it so that it will switch state if a key is pressed, the left mouse button has been pressed, or the screen is tapped. Replace the title screen with the following code.

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using SharedProject.Controls;

namespace SharedProject.GameScreens
{
    public interface ITitleState
    {
        GameState Tag { get; }
    }

    public class TitleState : GameState, ITitleState
    {
        #region Field region

        Texture2D backgroundImage;
        RenderTarget2D renderTarget;
```

```
        readonly GameState tag;

        public GameState Tag
        {
            get { return tag; }
        }

        #endregion

        #region Constructor region

        public TitleState(Game game)
            : base(game)
        {
            Game.Services.AddService<ITitleState>(this);
            SpriteBatch = Game.Services.GetService<SpriteBatch>();
        }

        #endregion

        #region XNA Method region

        protected override void LoadContent()
        {
            base.LoadContent();

            renderTarget = new(GraphicsDevice, Settings.BaseWidth, Settings.BaseHeight);

            ContentManager Content = Game.Content;
            backgroundImage = Content.Load<Texture2D>(@"Backgrounds\titlescreen");

            Label startLabel = new()
            {
                Position = new Vector2(350, 600),
                Text = "Tap to begin",
                Color = Color.White,
                TabStop = true,
                HasFocus = true
            };

            startLabel.Selected += StartLabel_Selected; ;

            ControlManager.Add(startLabel);
        }

        private void StartLabel_Selected(object sender, EventArgs e)
        {
            StartMenuState state = (StartMenuState)Game.Services.GetService<IStartMen-
uState>().Tag;
            StateManager.ChangeState(state);
        }

        public override void Update(GameTime gameTime)
        {
            if (Xin.WasMouseReleased(MouseButtons.Left) || Xin.TouchReleased() ||
Xin.WasKeyPressed())
            {
                StartLabel_Selected(this, null);
            }
            base.Update(gameTime);
```

```
        }

        public override void Draw(GameTime gameTime)
        {
            GraphicsDevice.SetRenderTarget(renderTarget);
            renderTarget.GraphicsDevice.Clear(Color.Black);

            SpriteBatch.Begin();

            SpriteBatch.Draw(
                backgroundImage,
                Settings.BaseRectangle,
                Color.White);

            base.Draw(gameTime);

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin();

            SpriteBatch.Draw(renderTarget, Settings.TargetRectangle, Color.White);

            SpriteBatch.End();
        }

        protected override void Show()
        {
            base.Show();

            LoadContent();
        }

        #endregion
    }
}
```

What has changed is that instead of creating a LinkLabel I create a regular label, setting it's text property to Tap to Begin. In the Update method, I check to see if the left mouse button has been pressed, the screen has been tapped, or any key has been pressed. If any of those three conditions are true, I call the StartLabel_Selected method to trigger changing the state.

Initially, I had planned on implementing the start menu. I think that this tutorial is long enough though and I don't want to start a new topic at this point. I don't want you to have too much to digest at once. I encourage you to visit the news page of my site, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials.

Good luck with your game programming adventures!
Cynthia