Lab 01: Touch in Android

Prerequisites

You will need a development environment, either a Mac or Windows PC with the Android SDK and Xamarin tools installed. We will be using the Android emulator to test the code we are building, so make sure to have a virtual device already configured and ready to run. See the **Xamarin.Android** setup documentation if you need help getting your environment setup:

http://docs.xamarin.com/guides/android/getting_started/installation/

Assets

This lab has several starter projects you will work with, all of these projects can be found at:

Content/Exercises/TouchAndroid/Lab Resources

Lab Goals

The goal of this lab will be to introduce touch on the Android platform. We will cover how to perform simple touch interaction, use built-in gesture recognition and finally create custom gestures.

The code samples have been split into multiple Activities for this solution with each Activity introducing a new concept:

- **TouchActivity** – Introduces simple touch and touch events
- **GestureRecognizerActivity** – Create a view that implements gesture recognizers such as the ScaleGestureDetector
- **CustomGestureRecognizerActivity** – Create a custom gesture using the Android Emulator and use a GestureOverlayView to determine which gesture was performed

The code introduced in this chapter is already present in the demo projects, but it's commented out.

The tasks for this lab are marked with a `// TODO:` task identifier and can be found quickly using the Tasks pane in Xamarin Studio or Visual Studio without having to navigate to individual files.

Steps

Open the Starting Solution

1. Launch Xamarin Studio.

Copyright © 2014 Xamarin, All Rights Reserved.

2. Open the **Touch_Android_Begin\Touch.Droid.sln** included in the course materials.

> Review Android Design Guidelines for more information on creating touchable user-interface components
>
> https://developer.android.com/design/style/metrics-grids.html

Simple Touch

1. Review the `Resources\Layout\touch_layout.axml` to get familiar with the user-interface for this lab
2. Open `Activities\TouchActivity.cs`
3. The first thing that we want to do is to enable our ImageView to be touched. Navigate to `// TODO: Step 1 - Listen for touch` events and uncomment the commented code below it.

```
_touchMeImageView.Touch += TouchMeImageViewOnTouch;
```

4. With the new touch listener created, let's start listening to events in the `TouchMeImageViewOnTouch` method. Navigate to `//TODO: Step 2 - Detect different touch event actions` events and uncomment the commented code below it.

```
switch (touchEventArgs.Event.Action & MotionEventActions.Mask)
{
    case MotionEventActions.Down:
        message = "Touch Down";
        break;
    case MotionEventActions.Move:
        message = "Touch Move";
        break;
    case MotionEventActions.Up:
        message = "Touch Ends";
        break;
    default:
        message = string.Empty;
        break;
}
```

5. The `Event` property of the `touchEventArgs` parameter provides a way to determine how touch events were performed on the View. It can also be used to see additional information such as touch duration and location as shown below.

Copyright © 2014 Xamarin, All Rights Reserved.

```
_durationInfoTextView.Text = String.Format ("Duration:  {0}", touchEve
ntArgs.Event.EventTime - touchEventArgs.Event.DownTime);


_coordinatesInfoTextView.Text = String.Format ("X: {0} Y: {1}", touchE
ventArgs.Event.GetX(), touchEventArgs.Event.GetY());
```

6. Run the application. Using the navigation, select the **Touch Simple** option. Interact with the "Touch Me" image and the interface will update with touch information.

Gesture Recognizer

1. Open the `Views/GestureRecognizerView.cs` file. This view will implement a scale gesture that we can use to update the position and size of an image on the screen. A scale gesture is commonly referred to as "pinch-to-zoom".
2. The first thing that we will need to do is to create a `ScaleGestureDetector`. Navigate to `//TODO: Step 3 - Create a subscription to scale gestures` events and uncomment the commented code below it

```
_scaleDetector = new ScaleGestureDetector(context, new MyScaleListener
(this));
```

3. The ScaleGestureDetector needs an IOnScaleGestureListener as the second parameter to detect and provide an implementation for when a scaling gesture is performed.
4. We will use a custom class named `MyScaleListener` that implements the `ScaleGestureDetector.SimpleOnScaleGestureListener`. The `SimpleOnScaleGestureListener` will satisfy our need for the `IOnScaleGestureListener` interface. This class will take in an instance of our `GestureRecognizerView` and update the view's `_scaleFactor` based on the detected scale gesture.
   Navigate to `//TODO: Step 4 - Create a scale gesture recognizer` events and uncomment the commented code below it

```
private readonly GestureRecognizerView _view;


public MyScaleListener(GestureRecognizerView view)
{
    _view = view;
}


public override bool OnScale(ScaleGestureDetector detector)
{
    _view._scaleFactor *= detector.ScaleFactor;
```

Copyright © 2014 Xamarin, All Rights Reserved.

```
    // put a limit on how small or big the image can get.
    if (_view._scaleFactor > 5.0f)
        _view._scaleFactor = 5.0f;

    if (_view._scaleFactor < 0.1f)
        _view._scaleFactor = 0.1f;

    _view.Invalidate();
    return true;
}
```

5. We will need to send the touch events from our view to our `ScaleGestureDetector`, so we need to implement the `OnTouchEvent` method of our view. Navigate to `//TODO: Step 5 - Subscribe to touch events in the view` events and uncomment the commented code below it.

```
public override bool OnTouchEvent(MotionEvent ev)
{
    Log.Debug(GetType().FullName, "Number of touches: {0}", ev.Pointer
Count);

    Log.Debug(GetType().FullName, "Touch Type: {0}", ev.Action);

    //TODO: Step 6 - Detect scale events
    //_scaleDetector.OnTouchEvent(ev);
    //
    //MotionEventActions action = ev.Action & MotionEventActions.Mask;
    //int pointerIndex;
    //
    //switch (action)
    //{
    //    case MotionEventActions.Down:
    //        _lastTouchX = ev.GetX ();
    //        _lastTouchY = ev.GetY ();
    //        _activePointerId = ev.GetPointerId (0);
    //        break;
    //
    //    case MotionEventActions.Move:
    //        pointerIndex = ev.FindPointerIndex (_activePointerId);
```

Copyright © 2014 Xamarin, All Rights Reserved.

```
    //        float x = ev.GetX (pointerIndex);
    //        float y = ev.GetY (pointerIndex);
    //
    //        if (!_scaleDetector.IsInProgress) {
    //            // Only move the ScaleGestureDetector isn't already
processing a gesture.
    //            float deltaX = x - _lastTouchX;
    //            float deltaY = y - _lastTouchY;
    //            _posX += deltaX;
    //            _posY += deltaY;
    //            Invalidate ();
    //        }
    //
    //        _lastTouchX = x;
    //        _lastTouchY = y;
    //        break;
    //
    //    case MotionEventActions.Up:
    //    case MotionEventActions.Cancel:
    //        // This events occur when something cancels the gesture
(for example the
    //        // activity going in the background) or when the pointer
 has been lifted up.
    //        // We no longer need to keep track of the active pointer
.
    //        _activePointerId = InvalidPointerId;
    //        break;
    //
    //    case MotionEventActions.PointerUp:
    //        // We only want to update the last touch position if the
 the appropriate pointer
    //        // has been lifted off the screen.
    //        pointerIndex = (int)(ev.Action & MotionEventActions.Poin
terIndexMask) >> (int)MotionEventActions.PointerIndexShift;
    //        int pointerId = ev.GetPointerId (pointerIndex);
    //        if (pointerId == _activePointerId) {
    //            // This was our active pointer going up. Choose a ne
w
```

Copyright © 2014 Xamarin, All Rights Reserved.

```
    //            // action pointer and adjust accordingly
    //            int newPointerIndex = pointerIndex == 0 ? 1 : 0;
    //            _lastTouchX = ev.GetX (newPointerIndex);
    //            _lastTouchY = ev.GetY (newPointerIndex);
    //            _activePointerId = ev.GetPointerId (newPointerIndex)
;
    //        }
    //        break;
    //}


    //Determine if the image is within the touch area
    if (Math.Abs(_scaleFactor -
 1.0f) > 0.001) { width *= _scaleFactor; height *= _scaleFactor; }
    var rc = new RectF(_posX, _posY, _posX + width, _posY + height);
    float touchX = ev.GetX(), touchY = ev.GetY();
    bool contains = rc.Contains(touchX, touchY);
    Log.Debug(GetType().FullName, "{0} - ({1},{2}) -
 CONTAINS: {3}", rc, touchX, touchY, contains);


    return true;
}
```

6. Our touch events will be passed to the touch events in the `_scaleDetector.OnTouchEvent` method. We will also implement touch events from our view to provide a way to capture touch information in our interface. Navigate to `//TODO: Step 6 - Detect scale events` and uncomment the commented code below it

```
_scaleDetector.OnTouchEvent(ev);


MotionEventActions action = ev.Action & MotionEventActions.Mask;
int pointerIndex;


switch (action)
{
    case MotionEventActions.Down:
        _lastTouchX = ev.GetX();
        _lastTouchY = ev.GetY();
        _activePointerId = ev.GetPointerId(0);
        break;
```

Copyright © 2014 Xamarin, All Rights Reserved.

```
    case MotionEventActions.Move:
        pointerIndex = ev.FindPointerIndex(_activePointerId);
        float x = ev.GetX(pointerIndex);
        float y = ev.GetY(pointerIndex);

        if (!_scaleDetector.IsInProgress)
        {
            // Only move the ScaleGestureDetector isn't already proces
sing a gesture.
            float deltaX = x - _lastTouchX;
            float deltaY = y - _lastTouchY;
            _posX += deltaX;
            _posY += deltaY;
            Invalidate();
        }

        _lastTouchX = x;
        _lastTouchY = y;
        break;

    case MotionEventActions.Up:
    case MotionEventActions.Cancel:
        // This events occur when something cancels the gesture (for e
xample the
        // activity going in the background) or when the pointer has b
een lifted up.
        // We no longer need to keep track of the active pointer.
        _activePointerId = InvalidPointerId;
        break;

    case MotionEventActions.PointerUp:
        // We only want to update the last touch position if the the a
ppropriate pointer
        // has been lifted off the screen.
        pointerIndex = (int)(ev.Action & MotionEventActions.PointerInd
exMask) >> (int)MotionEventActions.PointerIndexShift;
        int pointerId = ev.GetPointerId(pointerIndex);
```

Copyright © 2014 Xamarin, All Rights Reserved.

```
        if (pointerId == _activePointerId)
        {
            // This was our active pointer going up. Choose a new
            // action pointer and adjust accordingly
            int newPointerIndex = pointerIndex == 0 ? 1 : 0;
            _lastTouchX = ev.GetX(newPointerIndex);
            _lastTouchY = ev.GetY(newPointerIndex);
            _activePointerId = ev.GetPointerId(newPointerIndex);
        }
        break;
}
```

7. Using the variables that we captured during our touch events, we can update our view. Navigate to `//TODO: Step 7 - Apply Gesture updates` and uncomment the commented code below it

```
canvas.Save();
canvas.Translate(_posX, _posY);
canvas.Scale(_scaleFactor, _scaleFactor);

_icon.Draw(canvas);

canvas.Restore();
```

8. Open `Activities\GestureRecognizerActivity.cs`
9. Finally, we need to add our `GestureRecognizerView` to our Activity. This view will be the only view on the Activity and fill up the entire screen. Navigate to `//TODO: Step 8 - Set GestureRecognizerView as content view` and uncomment the commented code below it

```
SetContentView(new GestureRecognizerView(this));
```

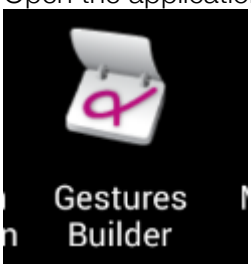
10. Run the application. Using the navigation, select the **Gesture Recognizer** option. Interact with the Activity using the "pinch" gesture to make the image on the screen larger or smaller.

Custom Gesture Recognizer

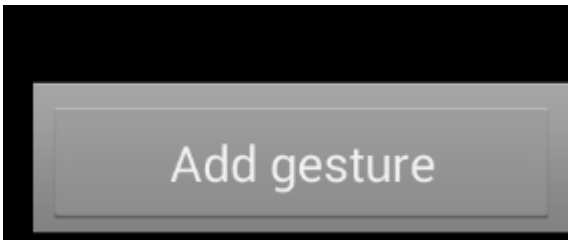
1. Start your Android emulator

Copyright © 2014 Xamarin, All Rights Reserved.

2. Open the application drawer and start the **Gesture Builder** application



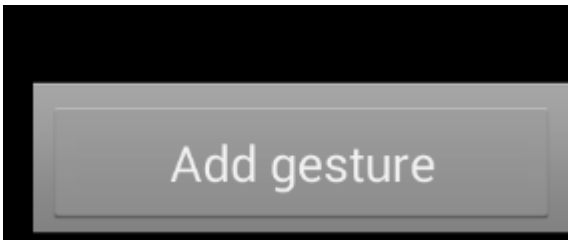
3. click the **Add Gesture** button



4. Set the name of the gesture to **checkmark** and draw a checkmark on the screen. Once finished, click the **done** button.

Copyright © 2014 Xamarin, All Rights Reserved.

5. click the **Add Gesture** button



6. Set the name of the gesture to **erase** and draw a back-and-forth scribble on the screen. Once finished, click the **done** button.

7. The gestures will be saved on the emulator in a file named gestures. In order to get those files off of the emulator, we will need to use a tool like adb or the Android Monitor tool to retrieve them.
   We already have gestures added to our project at `Resources\raw\gestures`

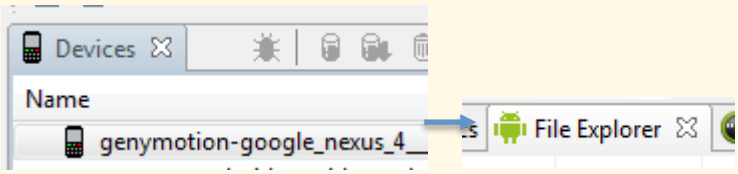
These files can be retrieved multiple ways. Below are two examples.

Using the adb tool, located in the android-sdk/platform-tools folder, run the following command to download the file to your project. Replace the <projectdirectory> tag with the actual path to your project.

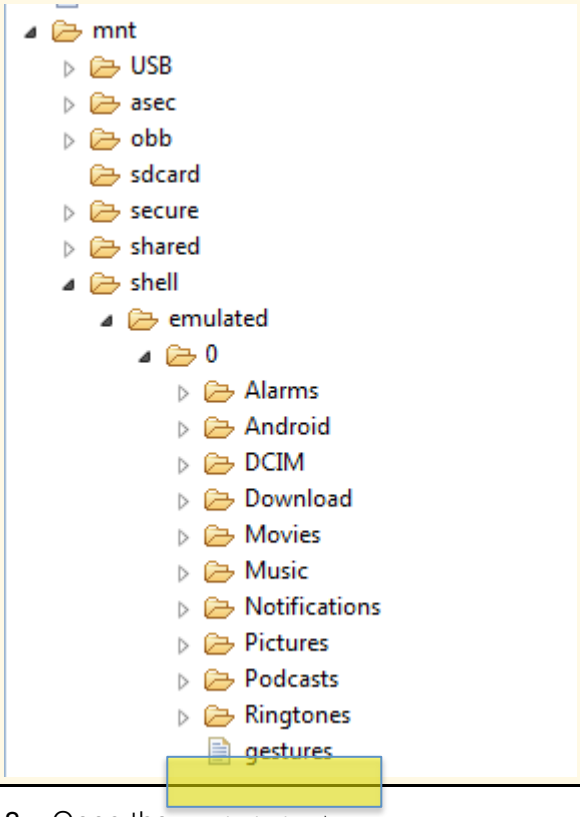
```
adb pull /storage/sdcard0/gestures <projectdirectory>/Resources/raw
```

Alternatively, you can use the **Android Debug Monitor** tool, available in the android-sdk/tools folder, to retrieve the file. With the application open, select your device from the **Devices** tab and then select the **File Explorer** tab.

Copyright © 2014 Xamarin, All Rights Reserved.



Using the File Explorer, navigate to the folder where your **gestures** file was saved.



8. Open the `Activities\CustomGestureRecognizerActivity.cs` class
9. Our Activity will to be able to detect gestures by setting the `ContentView` to a `GestureOVerlayView`. Navigate to `//TODO: Step 9 - Add a Gesture Overlay View as the primary view` and uncomment the commented code below it

```
// This activity will use a GestureOverlayView as it's content view. T
he layout file

// will be added as a subview of this.

GestureOverlayView gestureOverlayView = new GestureOverlayView(this);

SetContentView(gestureOverlayView);
```

10. The `GesturePerformed` event will provide us a way to determine which gesture has been performed and how to respond to it. Navigate to `//TODO: Step 10 - Subscribe to gesture events` and uncomment the commented code below it

```
gestureOverlayView.GesturePerformed += GestureOverlayViewOnGesturePerf
ormed;
```

Copyright © 2014 Xamarin, All Rights Reserved.

11. We will make our gestures available in our class using the `GestureLibraries.FromRawResource` method. Navigate to `//TODO: Step 11 - Load the gestures from the raw resource` and uncomment the commented code below it

```
// Load the binary gesture file that we created.
_gestureLibrary = GestureLibraries.FromRawResource(this, Resource.Raw.
gestures);
if (!_gestureLibrary.Load())
{
    Log.Error(GetType().FullName, "There was a problem loading the ges
ture library.");
    Finish();
}

// Load up the layout file for this activity and add it as child view
of the
// GestureOverlayView
View view = LayoutInflater.Inflate(Resource.Layout.custom_gesture_layo
ut, null);
_imageView = view.FindViewById<ImageView>(Resource.Id.imageView1);
gestureOverlayView.AddView(view);
```

12. Using the `GestureOverlayViewOnGesturePerformed` event handler, we can inspect the gesture that was performed. By providing our `gesturePerformedEventArgs.Gesture` property to the gesture library's `Recognize` method, we will receive a listing of potential gesture matches. Navigate to `//TODO: Step 12 - Get gestures that have a good decent score` and uncomment the commented code below it

```
IEnumerable<Prediction> predictions = from p in _gestureLibrary.Recogn
ize(gesturePerformedEventArgs.Gesture)
                                      orderby p.Score descending
                                      where p.Score > 1.0d
                                      select p;

Prediction prediction = predictions.FirstOrDefault();

if (prediction == null)
{
    Log.Debug(GetType().FullName, "Nothing seemed to match the user's
gesture, so don't do anything.");
    return;
```

Copyright © 2014 Xamarin, All Rights Reserved.

```
}


Log.Debug(GetType().FullName, "Using the prediction named {0} with a s
core of {1}.", prediction.Name, prediction.Score);
```

13. If we have good matches for our **checkmark** or **erase** gesture, update our image on the screen. Navigate to `//TODO: Step 13 - Determine which gesture was performed` and uncomment the commented code below it

```
if (prediction.Name.StartsWith("checkmark") && prediction.Score > 4.0f
 /* Use this to eliminate false positives */)
{
    _imageView.SetImageResource(Resource.Drawable.checked_me);
}
else if (prediction.Name.StartsWith("erase", StringComparison.OrdinalI
gnoreCase))
{
    // Match one of our "erase" gestures
    _imageView.SetImageResource(Resource.Drawable.check_me);
}
```

14. Run the application. Using the navigation, select the **Custom Gesture Recognizer** option. Interact with the Activity by performing the **checkmark** and **erase** gestures

Summary

In this lab, we learned how to integrate simple touch, gesture recognizers and custom gestures. . You can examine the completed version of this solution in the Touch_Android_Complete folder included in the lab resources.

Copyright © 2014 Xamarin, All Rights Reserved.