Lab 01 – Touch in iOS

Prerequisites

You will need a development environment, either a Mac or Windows PC with the either Xamarin Studio in OS X or Visual Studio with Xamarin tools installed. We will be using the iOS 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.iOS setup documentation if you need help getting your environment setup:

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

Downloads

Content/Exercises/TouchiOS/Part 01 Resources

Lab Goals

The goal of this lab will be to introduce touch on the iOS 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 ViewControllers for this solution with each ViewController introducing a new concept:

- SimpleTouch – Introduces simple touch and touch events
- GestureRecognizers – Create a view that implements gesture recognizers such as tapping and dragging
- CustomCheckmarkGestureRecognizer – Create a custom gesture that provides a way to determine if a checkmark gesture has been 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 in OS X or Visual Studio with the Xamarin.iOS tools installed.
2. Open the Touch_iOS_Begin\Touch.iOS.sln solution included in the course materials.

> Review Apple's Human Interface Guidelines for more information on creating touchable user-interface components
>
> https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/LayoutandAppearance.html

Simple Touch

1. Open the ViewControllers\SimpleTouch.cs class
2. We need to make sure that we can interact with our user interface. Set the following properties to true to ensure that both touch and multitouch are enabled for our ViewController's primary view. Locate the comment `// TODO: Step 1 - Ensure we can have user interaction` and uncomment the code below it.

```
View.UserInteractionEnabled = true;
View.MultipleTouchEnabled = true;
```

3. Our ViewController will have a method `TouchesBegan` that allows us to track when a new touch event occurred. Locate the comment `//TODO: Step 2 - Subscribe to touches began events` and uncomment the code below it.

```
public override void TouchesBegan(NSSet touches, UIEvent evt)
{
    base.TouchesBegan(touches, evt);

    // we can get the number of fingers from the touch count, but Multitouch must be enabled
    lblNumberOfFingers.Text = string.Format("Number of Fingers: {0}", touches.Count);

    // get the touch
    var touch = touches.AnyObject as UITouch;

    if (touch == null) return;

    Console.WriteLine("screen touched");

    //TODO: Step 3 - Check if touch was on a particular view
    //if (imgTouchMe.Frame.Contains(touch.LocationInView(View)))
    //    lblTouchStatus.Text = "Touch Status: Touches Began";

    //TODO: Step 4 - Detect multiple taps
```

```
    //if (touch.TapCount == 2 && imgTapMe.Frame.Contains(touch.Locatio
nInView(View)))
    //{
    //    imgTapMe.Image = UIImage.FromBundle(
    //        imageHighlighted
    //            ? "Images/DoubleTapMe.png"
    //            : "Images/DoubleTapMe_Highlighted.png");
    //
    //    imageHighlighted = !imageHighlighted;
    //}

    //TODO: Step 5 - Start recording a drag event
    //if (imgDragMe.Frame.Contains(touch.LocationInView(View)))
    //    touchStartedInside = true;

}
```

4. Using our UIImageView `imgTouchMe` we can pass our touch to the `Contains` method to determine if the touch was performed on that view or not. Locate the comment `//TODO: Step 3 - Check if touch was on a particular view` and uncomment the code below it.

```
if (imgTouchMe.Frame.Contains(touch.LocationInView(View)))
    lblTouchStatus.Text = "Touch Status: Touches Began";
```

Note: The touch variable calls the location `LocationInView` method. This method takes a view as a parameter and provides the coordinates of the touch in relation to that view.
When comparing two views for touch interaction, it is important to make sure that their touch coordinates are based around the same container view or your comparison will not be valid.

5. Using the `TapCount` property from the touch, we can determine if a double or more tap occurred. Next, we will use the `Touch` property along with our UIImageView `imgTapMe` to determine if that image was tapped twice. Locate the comment `//TODO: Step 4 - Detect multiple taps` and uncomment the code below it.

```
if (touch.TapCount == 2 && imgTapMe.Frame.Contains(touch.LocationInVie
w(View)))
{
    imgTapMe.Image = UIImage.FromBundle(
        imageHighlighted
```

```
            ? "Images/DoubleTapMe.png"
            : "Images/DoubleTapMe_Highlighted.png");

    imageHighlighted = !imageHighlighted;
}
```

6. The `TouchesBegan` method provides us a way to start recording touch events, such as drags, as well. Here we will set the `touchStartedInside` Boolean to `true` and use that variable to track our drag status. Locate the comment `//TODO: Step 5 - Start recording a drag event` and uncomment the code below it.

```
if (imgDragMe.Frame.Contains(touch.LocationInView(View)))
    touchStartedInside = true;
```

7. If our touch is cancelled, we will want to stop tracking our drag and we will set `touchStartedInside` to `false`. Locate the comment `//TODO: Step 6 - Listen for cancelled touches` and uncomment the code below it.

```
public override void TouchesCancelled(NSSet touches, UIEvent evt)
{
    base.TouchesCancelled(touches, evt);

    // reset our tracking flags
    touchStartedInside = false;
}
```

8. The `TouchesMoved` method will provide us a way to capture when a touch has begun and then moved. Locate the comment `//TODO: Step 7 - Listen for moved touches` and uncomment the code below it.

```
public override void TouchesMoved(NSSet touches, UIEvent evt)
{
    base.TouchesMoved(touches, evt);

    var touch = touches.AnyObject as UITouch;

    if (touch == null) return;

    if (imgTouchMe.Frame.Contains(touch.LocationInView(View)))
        lblTouchStatus.Text = "Touch Status: Touches Moved";
```

```
    ////TODO: Step 8 -
 If we started a drag event, update the frame for the image

    //if (!touchStartedInside) return;

    //

    //// move the shape

    //float offsetX = touch.PreviousLocationInView(View).X -
 touch.LocationInView(View).X;

    //float offsetY = touch.PreviousLocationInView(View).Y -
 touch.LocationInView(View).Y;

    //imgDragMe.Frame = new RectangleF(new PointF(imgDragMe.Frame.X -
 offsetX, imgDragMe.Frame.Y - offsetY), imgDragMe.Frame.Size);

}
```

9. If we are currently capturing our touch and drag using the `touchStartedInside` variable, then we can calculate the difference in movement using the touch's `PreviousLocationInView` and `LocationInView` methods to determine how much the user has moved. We will then move the image using this offset. Locate the comment `//TODO: Step 8 - If we started a drag event, update the frame for the image` and uncomment the code below it.

```
if (!touchStartedInside) return;


// move the shape

float offsetX = touch.PreviousLocationInView(View).X -
 touch.LocationInView(View).X;

float offsetY = touch.PreviousLocationInView(View).Y -
 touch.LocationInView(View).Y;

imgDragMe.Frame = new RectangleF(new PointF(imgDragMe.Frame.X -
 offsetX, imgDragMe.Frame.Y - offsetY), imgDragMe.Frame.Size);
```

10. The `TouchesEnded` method provides us a way to capture when our touch is complete and perform any additional cleanup for our tracking variables. Locate the comment `//TODO: Step 9 - Listen for ended touches` and uncomment the code below it.

```
public override void TouchesEnded(NSSet touches, UIEvent evt)

{

    base.TouchesEnded(touches, evt);


    var touch = touches.AnyObject as UITouch;


    if (touch == null) return;
```

```
    if (imgTouchMe.Frame.Contains(touch.LocationInView(View)))
        lblTouchStatus.Text = "Touch Status: Touches Ended";

    // reset our tracking flags
    touchStartedInside = false;
}
```

11. Run the application. Using the navigation, select the **Simple Touch** option. Interact with the images on the screen and the interface will update with touch information.

Gesture Recognizer

1. Open the `ViewControllers\GestureRecognizers.cs` class
2. In order to add a gesture recognizer to a view, we first need to create a `UIGestureRecognizer`. In this case, we will create a `UITapGestureRecognizer` that we will have listen for two (2) taps, set in the `NumberOfTapsRequired` property, before the gesture is executed. The `AddTarget` method allows us to provide a response for when the gesture is recognized. In this case, we will use the gesture to update a text label.
   Next, we will add it using the `AddGestureRecognizer` method of our view. Locate the comment `//TODO: Step 10 - Add a tap gesture recognizer` and uncomment the code below it.

```
var tapGestureRecognizer = new UITapGestureRecognizer();
tapGestureRecognizer.AddTarget(
    (tg) =>
    {
        var currTapGesture = (tg as UITapGestureRecognizer);
        if (currTapGesture == null) return;

        lblGestureStatus.Text = string.Format("Tap Location: @{0}", cu
rrTapGesture.LocationOfTouch(0, imgTapMe));
    });
tapGestureRecognizer.NumberOfTapsRequired = 2;

imgTapMe.AddGestureRecognizer(tapGestureRecognizer);
```

3. An alternative way to add create our handler for our gestures is to provide a method to handle when our gesture is detected. In this example, the `HandleDrag` method will be executed when our Pan/drag gesture is detected. Locate the comment `//TODO: Step 11a - Add a pan gesture recognizer for drag motions` and uncomment the code below it.

```
imgDragMe.AddGestureRecognizer(new UIPanGestureRecognizer(HandleDrag))
;
```

4. Depending on the state of the gesture, you may want to respond differently. In this example, if our drag is beginning, we will capture the original location of the `imgDragMe` UIImageView. If our touch is in a cancelled, failure or possible (start) state, we do not want to take any actions. Otherwise, we will use the touch to update the location of our `imgDragMe` UIImageView.
   Locate the comment `//TODO: Step 11b - Provide the handler drag event` and uncomment the code below it.

```
private void HandleDrag(UIPanGestureRecognizer recognizer)
{
    switch (recognizer.State)
    {
        case UIGestureRecognizerState.Began:
            // if it's just began, cache the location of the image
            originalImageFrame = imgDragMe.Frame;
            break;

        case UIGestureRecognizerState.Possible:
        case UIGestureRecognizerState.Cancelled:
        case UIGestureRecognizerState.Failed:
            return;
    }

    // move the shape by adding the offset to the object's frame
    PointF offset = recognizer.TranslationInView(imgDragMe);
    var newFrame = originalImageFrame;
    newFrame.Offset(offset.X, offset.Y);
    imgDragMe.Frame = newFrame;
}
```

5. Run the application. Using the navigation, select the **Gesture Recognizer** option. Double-tap and drag the images on the screen and the interface will update with touch information.

Custom Gesture Recognizer

1. Open `GestureRecognizer\CheckmarkGestureRecognizer.cs`
2. To create a custom gesture recognizer, we need to create a view with a base class of `UIGestureRecognizer`. This class will need to use our different touch methods to determine when we have performed a gesture properly. In this case, we will be detecting a check gesture.

Locate the comment `//TODO: Step 12 - Create a custom Checkmark Gesture Recognizer` and uncomment the code below it.

```
public class CheckmarkGestureRecognizer : UIGestureRecognizer
{
    // declarations
    protected PointF midpoint = PointF.Empty;
    protected bool strokeUp = false;

    /// <summary>
    ///   Called when the touches end or the recognizer state fails
    /// </summary>
    public override void Reset()
    {
        base.Reset();

        strokeUp = false;
        midpoint = PointF.Empty;
    }

    /// <summary>
    ///   Is called when the fingers touch the screen.
    /// </summary>
    public override void TouchesBegan(NSSet touches, UIEvent evt)
    {
        base.TouchesBegan(touches, evt);

        // we want one and only one finger
        if (touches.Count != 1)
            State = UIGestureRecognizerState.Failed;

        Console.WriteLine(State.ToString());
    }

    /// <summary>
    ///   Called when the touches are cancelled due to a phone call, e
tc.
```

```
    /// </summary>
    public override void TouchesCancelled(NSSet touches, UIEvent evt)
    {
        base.TouchesCancelled(touches, evt);
        // we fail the recognizer so that there isn't unexpected behav
ior
        // if the application comes back into view
        State = UIGestureRecognizerState.Failed;
    }

    /// <summary>
    ///   Called when the fingers lift off the screen
    /// </summary>
    public override void TouchesEnded(NSSet touches, UIEvent evt)
    {
        base.TouchesEnded(touches, evt);
        //
        if (State == UIGestureRecognizerState.Possible && strokeUp)
        {
            State = UIGestureRecognizerState.Recognized;
        }

      midpoint = PointF.Empty;

      Console.WriteLine(State.ToString());
    }

    /// <summary>
    ///   Called when the fingers move
    /// </summary>
    public override void TouchesMoved(NSSet touches, UIEvent evt)
    {
        base.TouchesMoved(touches, evt);

        // if we haven't already failed
        if (State != UIGestureRecognizerState.Failed)
```

```
        {
            // get the current and previous touch point
            var newPoint = (touches.AnyObject as UITouch).LocationInVi
ew(View);
            var previousPoint = (touches.AnyObject as UITouch).Previou
sLocationInView(View);

            // if we're not already on the upstroke
            if (!strokeUp)
            {
                // if we're moving down, just continue to set the midp
oint at
                // whatever point we're at. when we start to stroke up
, it'll stick
                // as the last point before we upticked
                if (newPoint.X >= previousPoint.X && newPoint.Y >= pre
viousPoint.Y)
                {
                    midpoint = newPoint;
                }
                    // if we're stroking up (moving right x and up y [
y axis is flipped])
                else if (newPoint.X >= previousPoint.X && newPoint.Y <
= previousPoint.Y && midpoint != PointF.Empty)
                {
                    strokeUp = true;
                }
                    // otherwise, we fail the recognizer
                else
                {
                    State = UIGestureRecognizerState.Failed;
                }
            }
        }

        Console.WriteLine(State.ToString());
    }
}
```

3. Open `ViewControllers\CustomCheckmarkGestureRecognizer.cs`
4. Once we have our custom gesture created, we will need to assign it to a view. In this case, we will have our primary view for our view controller listen for the checkmark gesture and update the displayed image, if we have a match. Locate the comment `//TODO: Step 13 - Listen for the Checkmark gesture` and uncomment the code below it.

```
protected void WireUpCheckmarkGestureRecognizer()
{
    // create the recognizer
    checkmarkGesture = new CheckmarkGestureRecognizer();

    // wire up the event handler
    checkmarkGesture.AddTarget(() =>
    {
        //TODO: Step 14 - Check the state of the gesture
        if (checkmarkGesture.State != UIGestureRecognizerState.Recogni
zed ||
            checkmarkGesture.State != UIGestureRecognizerState.Ended)
            return;

        BeginInvokeOnMainThread(() =>
        {
            imgCheckmark.Image = UIImage.FromBundle(
                isChecked
                    ? "Images/CheckBox_Checked.png"
                    : "Images/CheckBox_Unchecked.png");
        });

        isChecked = !isChecked;
    });

    //TODO: Step 15 - Add the gesture to the view
    View.AddGestureRecognizer(checkmarkGesture);
}
```

5. If our gesture has been recognized or finished, update the displayed image. Locate the comment `//TODO: Step 14 - Check the state of the gesture` and uncomment the code below it.

```
if (checkmarkGesture.State != UIGestureRecognizerState.Recognized ||
    checkmarkGesture.State != UIGestureRecognizerState.Ended)
    return;

BeginInvokeOnMainThread(() =>
{
    imgCheckmark.Image = UIImage.FromBundle(
        isChecked
            ? "Images/CheckBox_Checked.png"
            : "Images/CheckBox_Unchecked.png");
});

isChecked = !isChecked;
```

6. Add the gesture to the view using the `AddGestureRecognizer` method. Locate the comment `//TODO: Step 15 - Add the gesture to the view` and uncomment the code below it.

```
View.AddGestureRecognizer(checkmarkGesture);
```

7. Run the application. Using the navigation, select the **Custom Gesture Recognizer** option. Draw a checkmark on the screen and the image will update, if there is a match.

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_iOS_Complete folder included in the lab resources.