IOS240

Touch and Gestures

Download class materials from
university.xamarin.com

Microsoft Xamarin University

Objectives

1. Respond to touch events
2. Handle multi-touch events
3. Utilize gestures

Respond to touch events

Xamarin University

Tasks

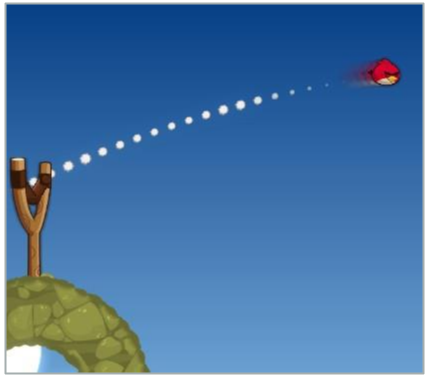
1. Enabling touch events on a view
2. Responding to touch events

Motivation

- Touch-based UIs have become the expected way to interact with electronic devices through standardized gestures

Maps and navigation

Books and magazines



... and of course games!

Touch in iOS

- There are two ways to programmatically respond to iOS touch interactions

Touch Events

Low-level events

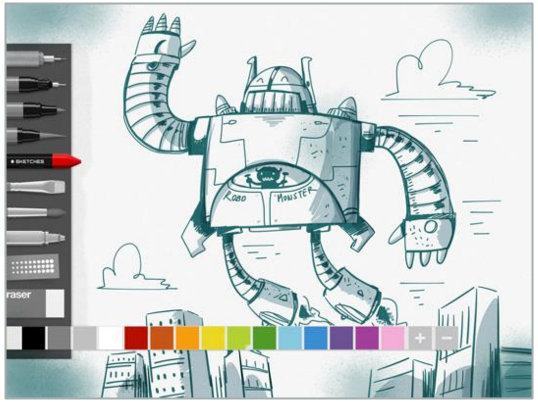
Gestures

High-level actions

Touch events

- Utilize low-level touch events when complete control over touch interactions is required

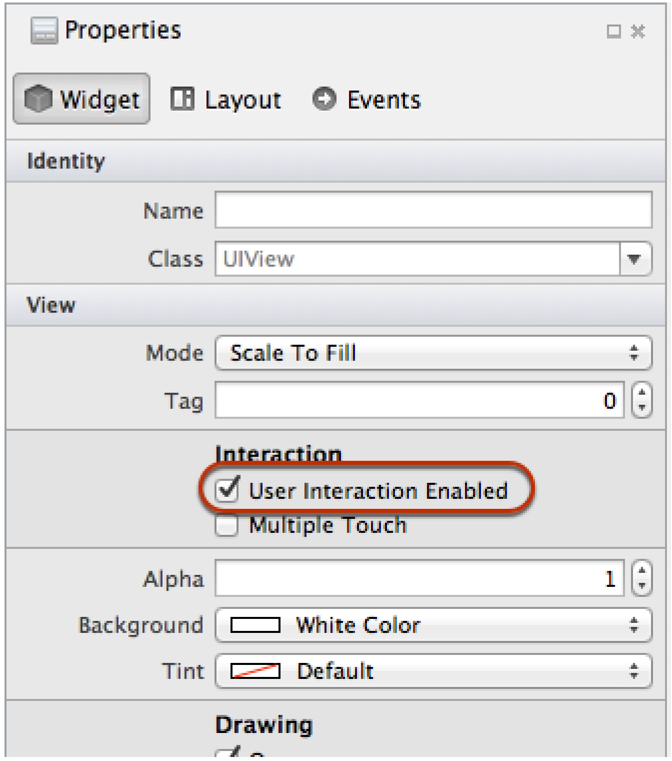
Games



Creative Apps

Enable touch in the designer

- We enable touch interactions for a selected view in the designer by checking User Interaction Enabled


Properties

Widget | Layout | Events

Identity

Name	
Class	UIView

View

Mode	Scale To Fill
Tag	0

Interaction

- [✓] User Interaction Enabled
- [ ] Multiple Touch

Alpha	1
Background	White Color
Tint	Default

Drawing

Enable touch programmatically

- Enable touch programmatically by setting the `UserInteractionEnabled` property on a `UIView` to `true`

```
var myImageView = new UIImageView (...);
myImageView.UserInteractionEnabled = true;
```

↑

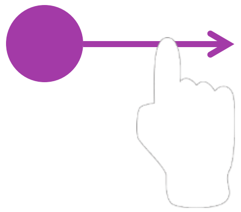
If this property is `false`, touch events will fall through to the parent

Some views have `UserInteractionEnabled` by default, but it's generally a good idea to set it explicitly

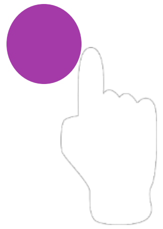
What are touch events?

- **Touch events** are low level interactions which are reported by a **`UIView`** and include:

Began



Moved



Ended

Cancelled

Touch event methods

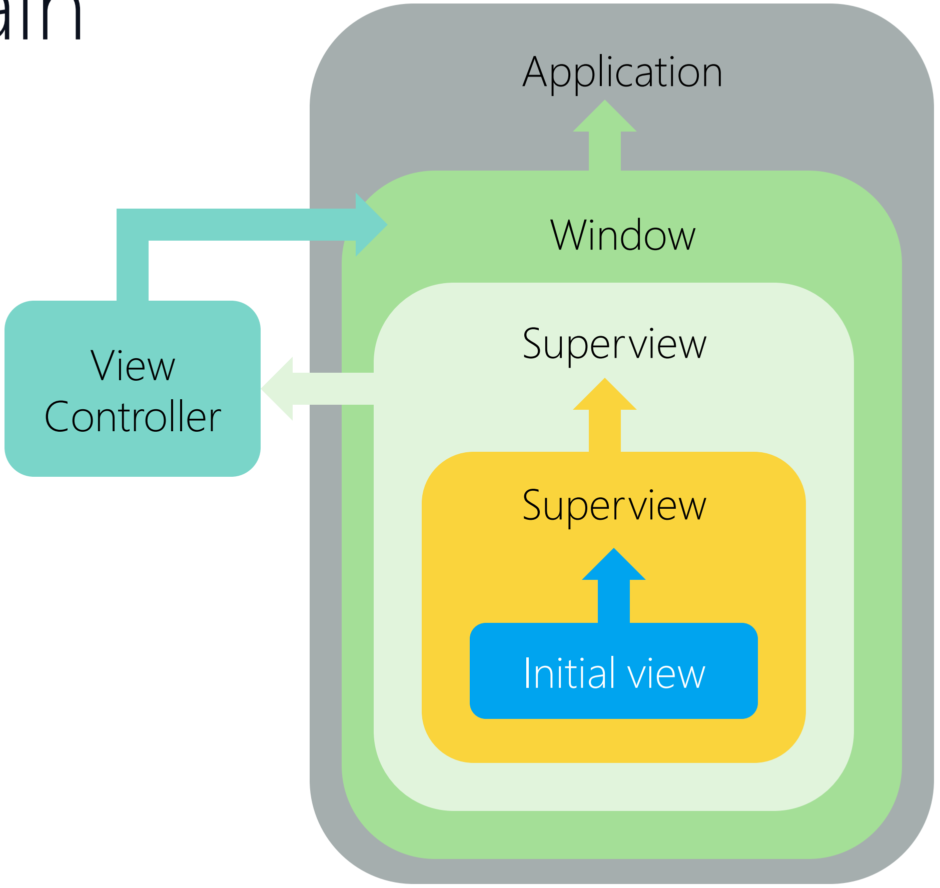
- There are four phases to a touch interaction, each represented by a virtual method on the base **`UIResponder`** class which is inherited by **`UIView`**, **`UIViewController`** and **`UIApplication`**

```
public class UIResponder
{
    ...
    public virtual void TouchesBegan (NSSet touches, ... ) {...}
    public virtual void TouchesMoved (NSSet touches, ... ) {...}
    public virtual void TouchesEnded (NSSet touches, ... ) {...}
    public virtual void TouchesCancelled (NSSet touches, ... ){...}
}
```

The Responder Chain

- iOS walks the responder chain of **`UIResponder`** objects to find a handler for a touch event

- System performs hit-testing to identify the initial view (called the *first responder*) and then continues up the view hierarchy



Group Exercise

Override Touch Methods

Xamarin University

Touch data within NSSet

- Override the touch event handler methods to receive data for each phase of a finger touch via the **`NSSet`** touches argument

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

    ...
}
```

`NSSet` contains specific information about the touch event

Retrieving the touch data

- **NSSet** touches holds **UITouch** objects representing each finger interacting with the screen

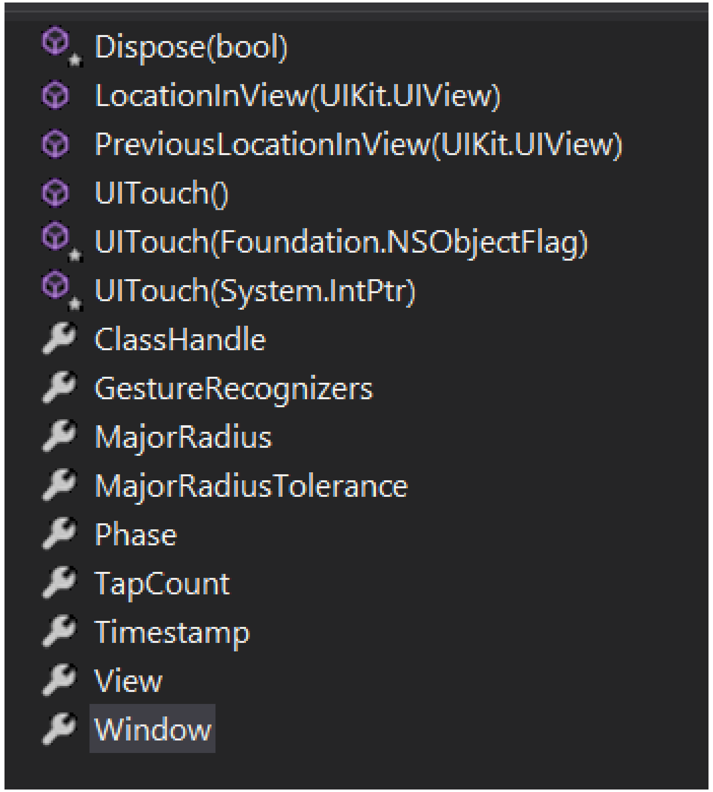
```
public override void TouchesMoved(NSSet touches, UIEvent evt)
{
   base.TouchesMoved(touches, evt);

   var touch = touches.AnyObject as UITouch;

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

Then cast it to a **UITouch** object

Use the **AnyObject** property to obtain the first **UITouch**

What is UITouch?

- A **`UITouch`** object contains the data representing the presence or movement of a finger onscreen including:
  - Current Location
  - Previous Location
  - Phase
  - Tap Count
  - Pressure (via radius)
  - etc.



```
Dispose(bool)
LocationInView(UIKit.UIView)
PreviousLocationInView(UIKit.UIView)
UITouch()
UITouch(Foundation.NSObjectFlag)
UITouch(System.IntPtr)
ClassHandle
GestureRecognizers
MajorRadius
MajorRadiusTolerance
Phase
TapCount
Timestamp
View
Window
```

Xamarin University

Touch location

❖ We can get the location of the current touch event from using the `LocationInView` method

```
public override void TouchesMoved(NSSet touches, UIEvent evt)
{
    var touch = touches.AnyObject as UITouch;

    nfloat xPos = touch.LocationInView(this.View).X;
    nfloat yPos = touch.LocationInView(this.View).Y;
    ...
}
```

Xamarin University

Touch movement

❖ Get the location of the current touch event from the **`LocationInView`** method - To find the movement delta, we also use **`PreviousLocationInView`**

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

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

Returns the previously reported location of the view

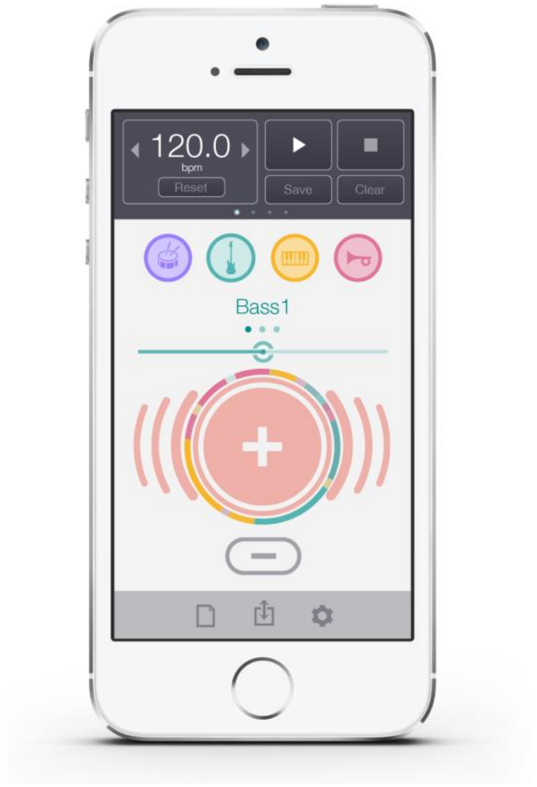
Individual Exercise

Drag and Snap

Xamarin University

Provide visual cues for touchable UI

- Visual cues help the user understand where to interact with the screen, common cues include:
  - Color
  - Location
  - Context
  - Icons
  - Labels
  - Animations



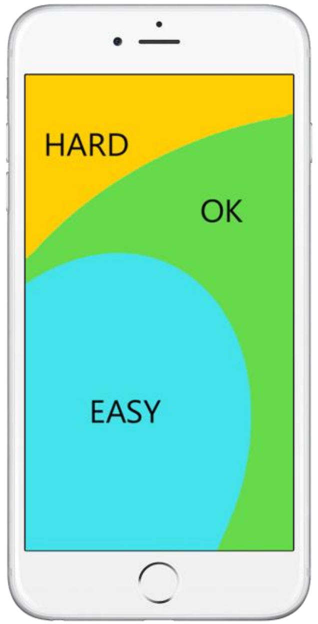
Sizing and spacing your UI

- Apple recommends a minimum of 44x44 points for touchable UI

  - 44 is the *minimum* – larger is ok
  - Leave white space between touchable UI

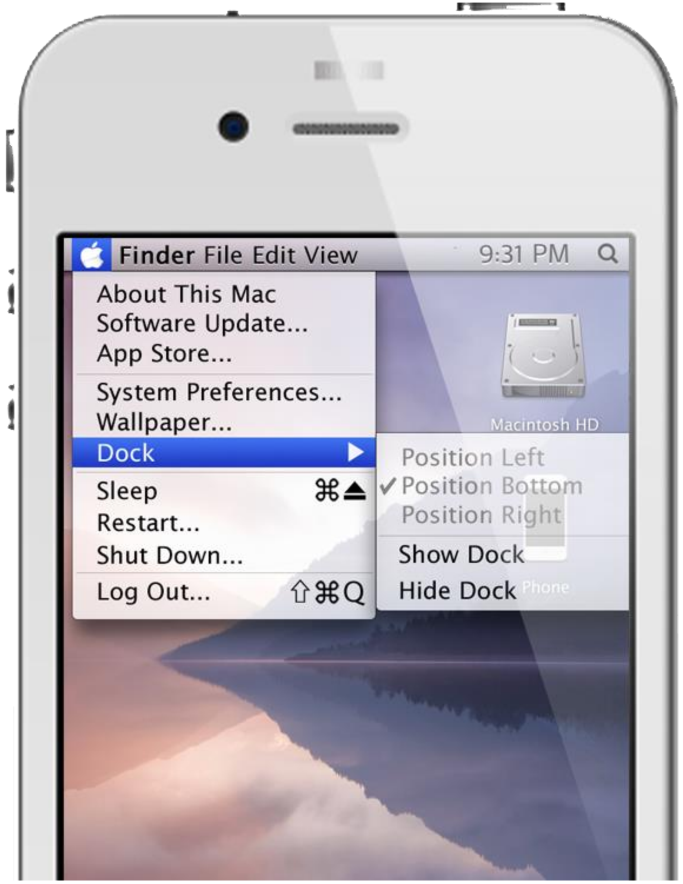
Design for fingers

- Consider how your user will hold the device and use your application, considerations include:
  - Common interaction scenarios
  - What parts of the screen may be obscured by the hand
  - Relative position of related controls



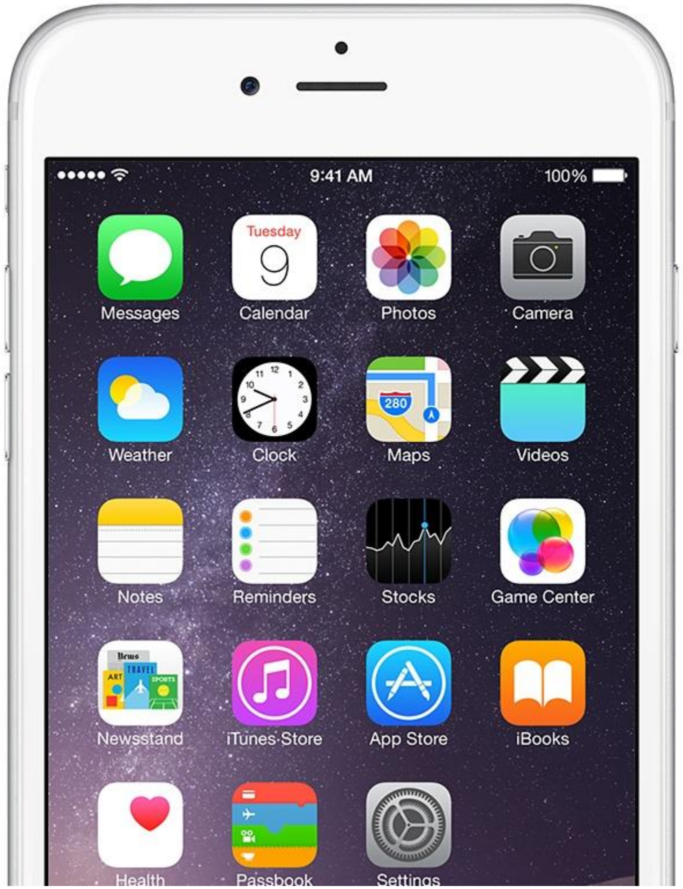
UI design

❖ What's wrong with this UI?



UI design

- Why does this UI work?



Flash Quiz

Flash Quiz

① You should always leave 44 points of space between views

   a) True

   b) False

Flash Quiz

① You should always leave 44 points of space between views

a) True

b) False

Flash Quiz

② What types of cues can you use to indicate interactivity?

a) Color

b) Location

c) Animations

d) All the above

Flash Quiz

② What types of cues can you use to indicate interactivity?

a) Color

b) Location

c) Animations

d) <u>All the above</u>

Summary

1. Enabling touch events on a view
2. Responding to touch events

Handle multi-touch events

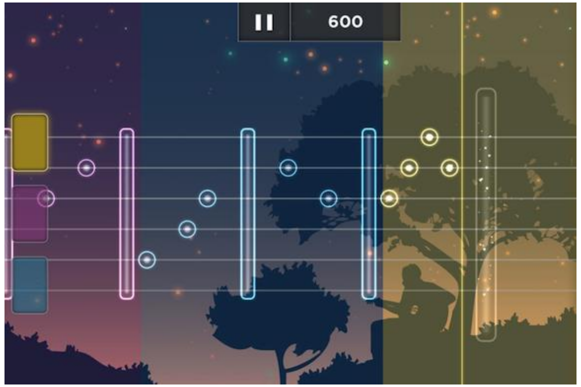
Xamarin University

Tasks

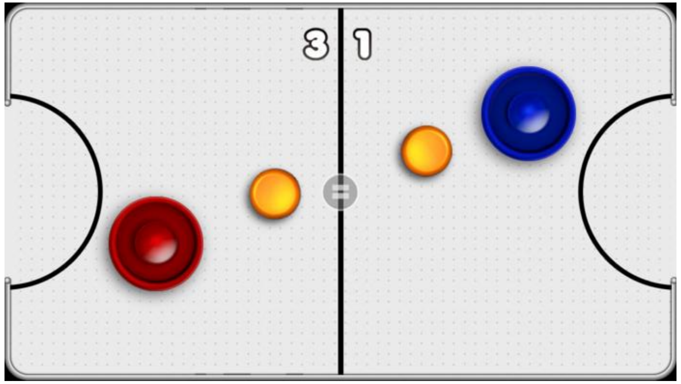
1. Enable multi-touch on views
2. Respond to multi-touch events

Multi-touch events

- Complex UI may need to track multiple fingers simultaneously – commonly used in entertainment applications



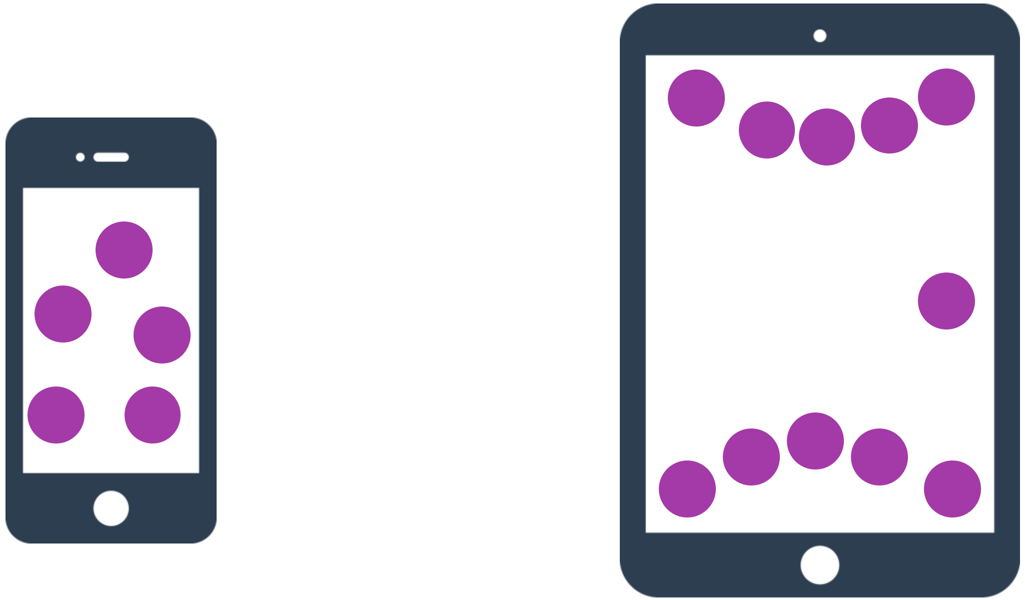
Music apps



Multiplayer or complex games

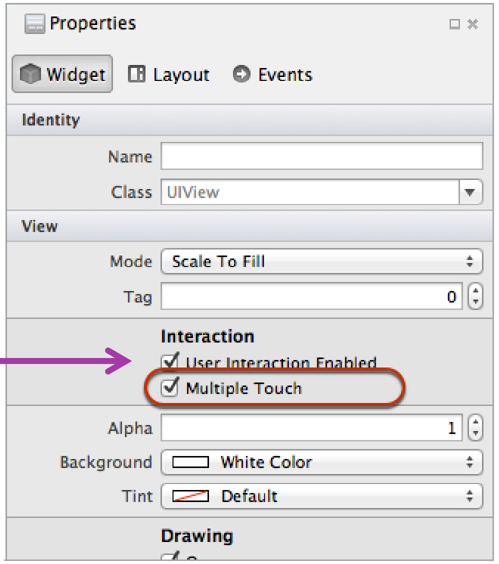
How many simultaneous touch points?

- Current iPhones are capable of tracking up to 5 simultaneous touch points; the iPad is capable of 11



Enabling multi-touch in the designer

- Multi-touch isn't enabled by default but can be enabled in the Designer by checking **Multiple Touch** in the properties pain for a selected view


Properties

Widget | Layout | Events

Identity

Name	
Class	UIView

View

Mode	Scale To Fill
Tag	0

Interaction

✓ User Interaction Enabled

✓ Multiple Touch

Alpha	1
Background	White Color
Tint	Default

Drawing

User Interaction Enabled
must also be checked

Enabling multi-touch programmatically

- Multi-touch can be enabled programmatically by setting the `MultipleTouchEnabled` property of a `UIView` to `true`

```
UIImageView myImageView = new UIImageView (...);

myImageView.UserInteractionEnabled = true;

myImageView.MultipleTouchEnabled = true;
```

User interaction must also be enabled

Xamarin University

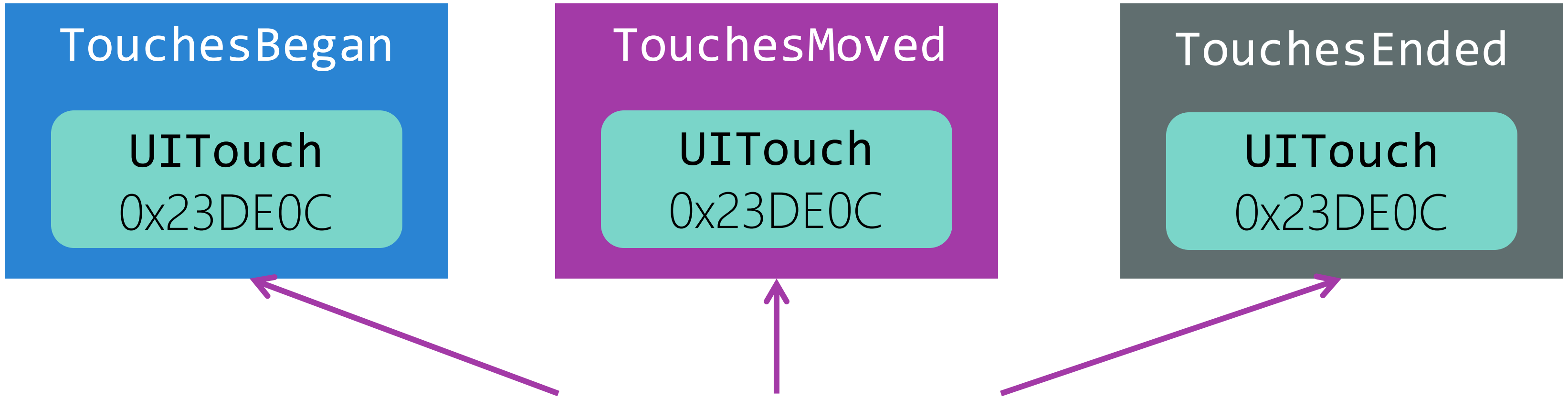
NSSet and multi-touch

❖ For multi-touch, iOS will pass in a **UITouch** object to the touch overrides for each finger who's phase has changed or properties have updated

```
public override void TouchesMoved(NSSet touches, UIEvent evt)
{
    foreach (UITouch touch in touches)
    {
        var loc = touch.LocationInView (this.View);
        ...
    }
}
```

UITouch and multi-touch

- **`UITouch`** objects persist across across touch phases for a given sequence of motions



The same object for a specific finger will be presented in each event method

Xamarin University

Tracking touch across phases

- The **Handle** for a specific **UITouch** object will persist across phases and can be used to reference other data

```
var colors = new Dictionary<IntPtr, UIColor>();
```

```
public override void TouchesBegan(NSSet touches, UIEvent evt)
{
   foreach (UITouch touch in touches) {
       colors.Add(touch.Handle, lineColor);
   }
   ...
}
```

Individual Exercise

Xam Paint

Xamarin University

Summary

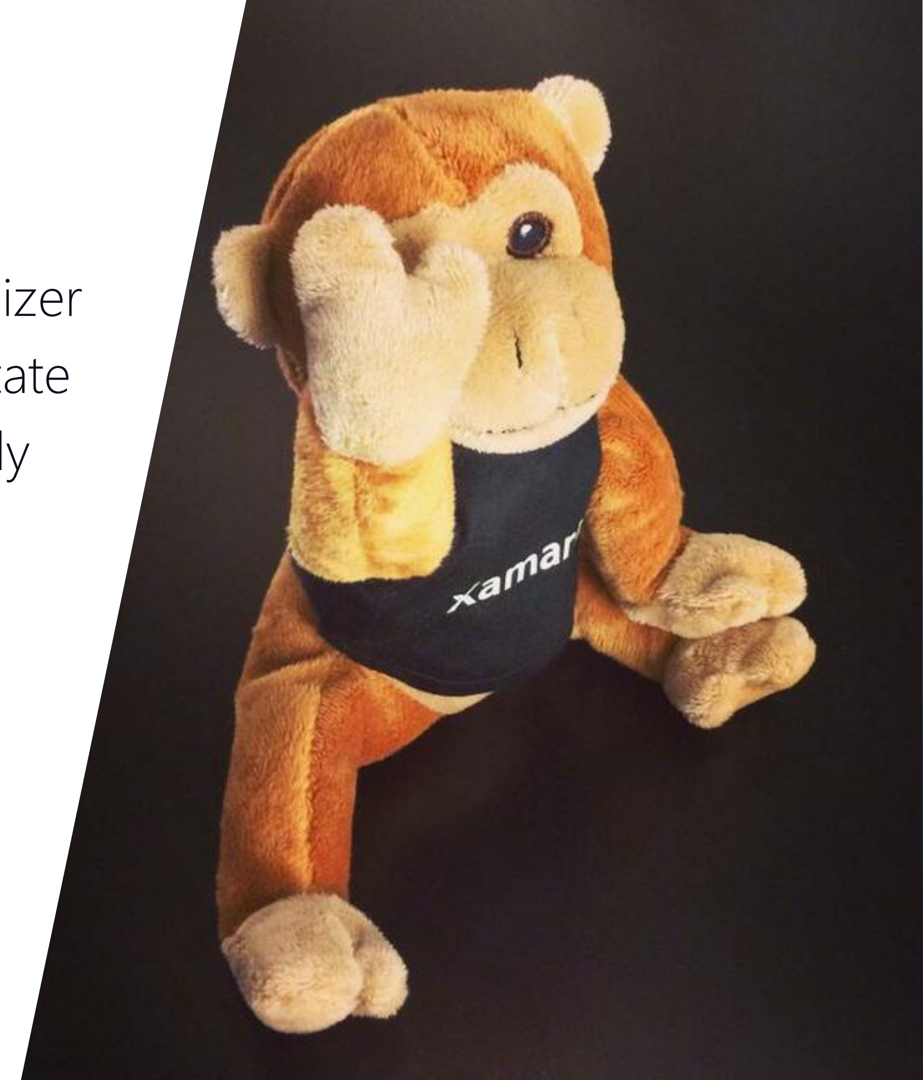
1. Enable multi-touch on views
2. Respond to multi-touch events

Implement gestures

Xamarin University

Tasks

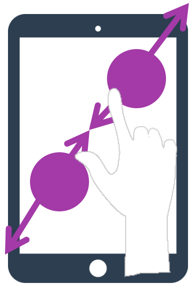
1. Create and assign a gesture recognizer
2. Respond to a gesture's change in state
3. Use multiple gestures simultaneously

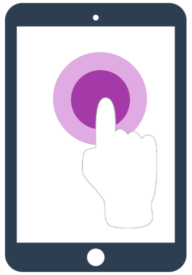


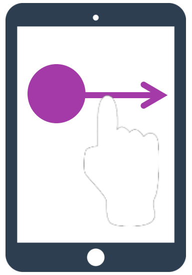
What are iOS gestures?

- *Gestures* are recognized as a continuous series of touch events performed by the user to invoke a specific task

Tap


Pinch


Long Press


Pan

Discrete and continuous gestures

- There are two types of gestures: discrete and continuous

Discrete gestures	contain one or more finite touch events such as tap
Continuous gestures	has no fixed path and may be carried out indefinitely, for example, pinch-and-zoom

What is UIGestureRecognizer

- `UIGestureRecognizer` converts low-level touch events into higher-level actions that correspond to discrete or continuous gestures

Discrete gestures	Tap UITapGestureRecognizer	Swipe UISwipeGestureRecognizer	Long Press UILongPressGestureRecognizer
Continuous gestures	Pinch UIPinchGestureRecognizer	Rotation UIRotationGestureRecognizer	Pan UIPanGestureRecognizer

Steps to use a gesture

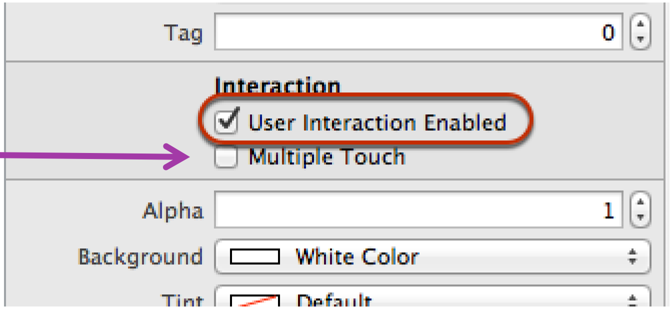
1) Enable user interaction
2) Create a gesture recognizer
3) Configure the gesture recognizer (if needed)
4) Set the target method to execute on completion
5) Add the gesture recognizer to a View

Enable user interaction [1]

- Set the **`UserInteractionEnabled`** property on a view to true to enable user interaction

```
myImageView.UserInteractionEnabled = true;
myImageView.MultipleTouchEnabled = true;
```

enable multi-touch for gestures that use more than one finger



Tag: 0

Interaction
- [✓] User Interaction Enabled
- [ ] Multiple Touch

Alpha: 1

Background: White Color

Tint: Default

Create a gesture recognizer [2]

- Choose and instantiate a gesture recognizer based on the type of gesture you're using

```
public override OnCreate ()
{
    var doubleTapGesture = new UITapGestureRecognizer ();
    ...
}
```

Xamarin University

Configure the gesture recognizer [3]

- Some gesture recognizers can be customized by using their public properties

```
var doubleTapGesture = new UITapGestureRecognizer ();

doubleTapGesture.NumberOfTapsRequired = 2;
...
```

Set the target [4]

- Use the **`AddTarget`** method to specify what **`Action`** should be raised when the gesture is performed

```
var doubleTapGesture = new UITapGestureRecognizer ();

...
doubleTapGesture.AddTarget (() => {
      Debug.WriteLine("double tap");
});
```

or

```
var doubleTapGesture = new UITapGestureRecognizer (OnTap);
```

Xamarin University

Add the gesture recognizer [5]

- The gesture recognizer must be added to **UIView** to receive touch events

```
myImageView = new UIImageView(...);
myImageView.UserInteractionEnabled = true;

var doubleTapGesture = new UITapGestureRecognizer ();

...

myImageView.AddGestureRecognizer (doubleTapGesture);
```

Responding to gestures

- The Action associated with a gesture recognizer's target can optionally receive the recognizer object which can be used to retrieve specific details about the gesture

```
var rotationGR = new UIRotationGestureRecognizer (OnRotation);
```

```
void OnRotation(UIRotationGestureRecognizer gesture)
{
   var currentRotation = gesture.Rotation;
   ...
}
```

Transforms

- The Core Graphics **`CGAffineTransform`** structure can be used to rotate, scale, and translate **`UIViews`**

```
CGAffineTransform transform = CGAffineTransform.MakeIdentity ();

transform.Rotate (angle: rotationInRadians);

transform.Scale (scaleX, scaleY);

transform.Translate (translateX, translate.Y);

myUIView.Transform = transform;
```

Factory method creates a transform using an identity matrix

`CGAffineTransform` exposes methods to rotate, scale, translate and skew

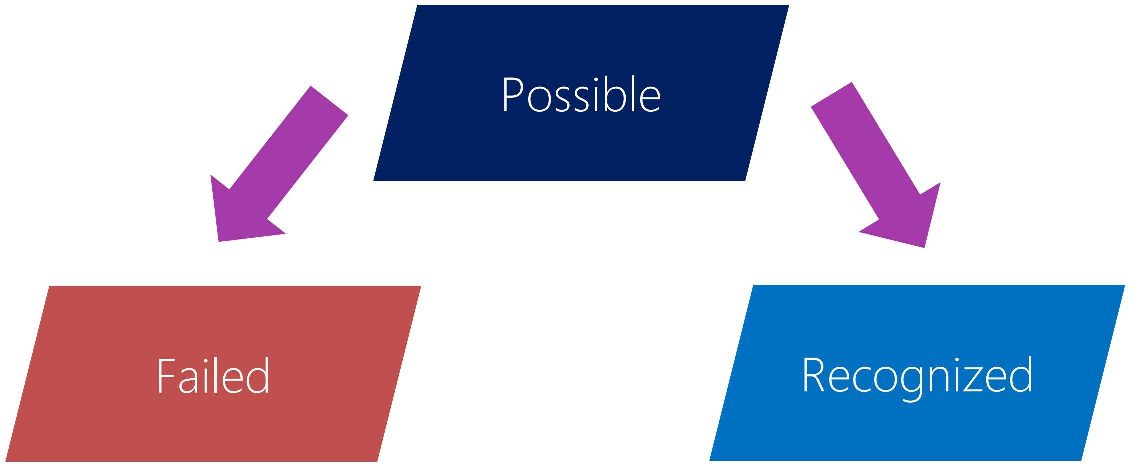
Apply the transform by assigning the Transform property of a UIView

Individual Exercise

Pan gestures

Discrete gesture recognizer states

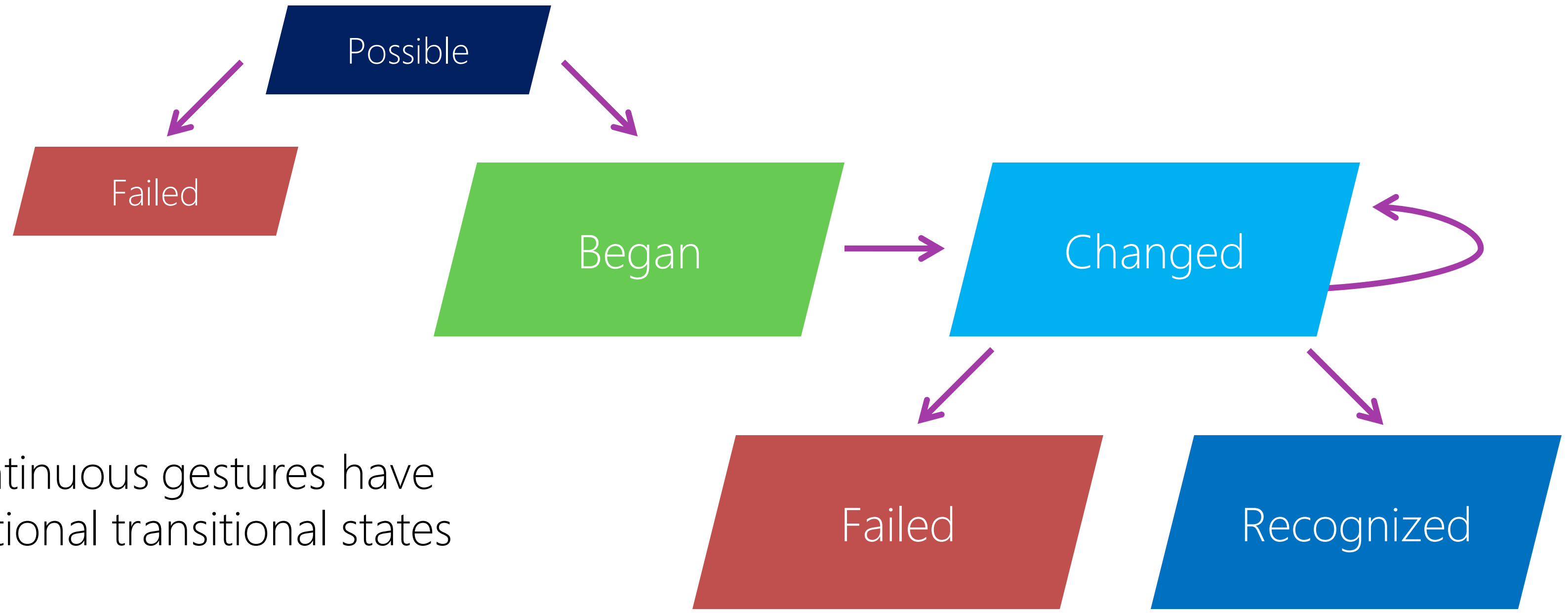
- Gesture recognizers transition through states in a predefined fashion



Discrete gestures can be in 1 of 3 states

Continuous gesture recognizer states

- Gesture recognizers transition from one state to another in a predefined way



Continuous gestures have additional transitional states

Recognizing gesture states

- The **`UIGestureRecognizer`**'s target will be called as the gesture changes states

```
void HandleRotation(UIRotationGestureRecognizer gesture)
{
   switch (gesture.State) {
   case UIGestureRecognizerState.Possible: break;
   case UIGestureRecognizerState.Began: break;
   case UIGestureRecognizerState.Recognized: break;
   case UIGestureRecognizerState.Changed: break;
   case UIGestureRecognizerState.Failed: break;
   ...
}
```

Xamarin University

Using gestures simultaneously

- Can use multiple gesture recognizers together, but must enable support in code on each recognizer

```
rotationGesture.ShouldRecognizeSimultaneously = IsSimultaneous;
```

```
swipeGesture.ShouldRecognizeSimultaneously = IsSimultaneous;
```

```
public bool IsSimultaneous (UIGestureRecognizer gestureRecognizer,
   UIGestureRecognizer otherGestureRecognizer)
{
   return ShouldAllowGesture (otherGestureRocognizer);
}
```

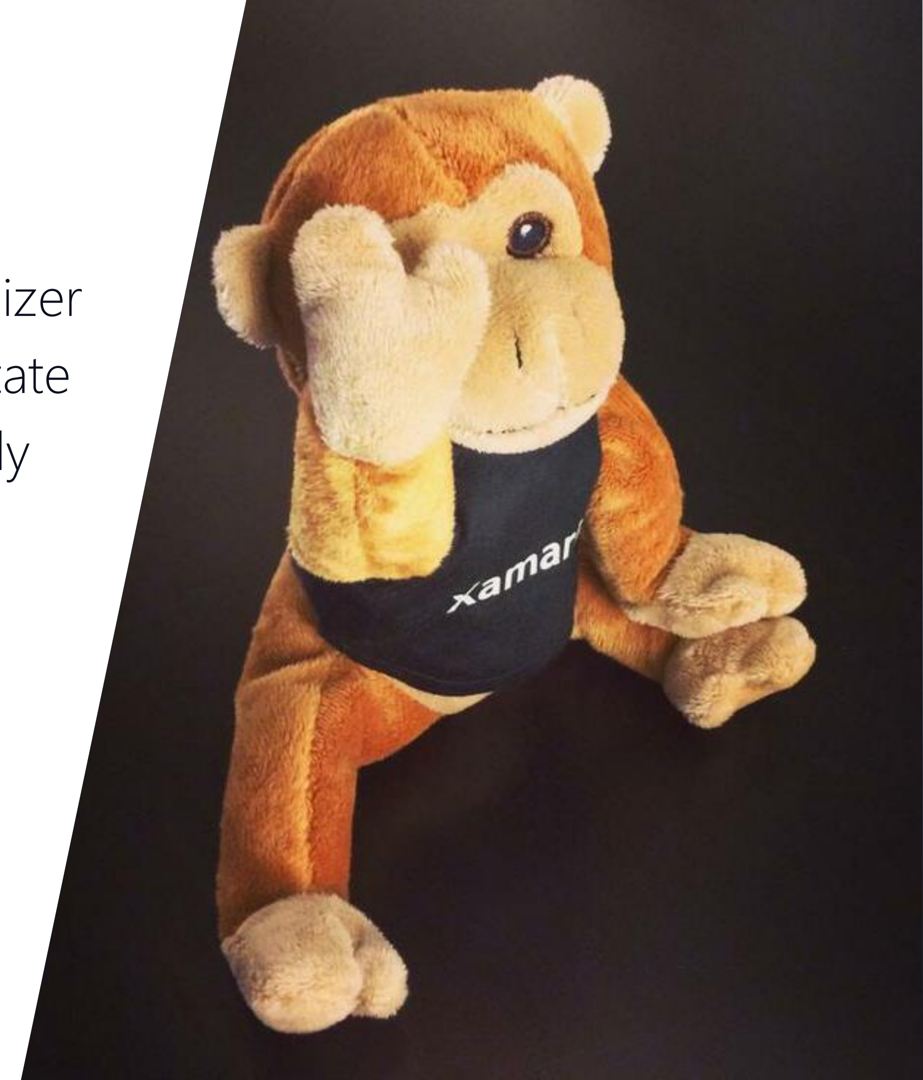
Individual Exercise

Using Multiple gestures – add scale and rotation

Xamarin University

Summary

1. Create and assign a gesture recognizer
2. Respond to a gesture's change in state
3. Use multiple gestures simultaneously



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft