06016323 Mobile Device Programming

CHAPTER 13 : GESTURE

Gesture

- Gesture : การควบคุมแอปพลิเคชันด้วยการเคลื่อนไหวของมือ
- Gesture มักใช้ร่วมกับ Animation เพื่อทำให้แอปพลิเคชันตอบสนองกับผู้ใช้ได้ดีลำให้ใช้งานได้ง่าย
- ตัวอย่างของ Gesture อย่างง่าย เช่น TouchableOpacity หรือ TouchableHilight เป็นต้น (กรณีที่มีการแตะ-tap)

Gesture Responder System

- ทำหน้าที่จัดการ lifecycle ของ gesture ในแอปพลิเคชัน
- ต้องพิจารณาได้ว่าการสัมผัสหน้าจอของผู้ใช้เป็นรูปแบบใด เช่น scrolling, sliding หรือ tapping เป็นต้น
- จัดเตรียม callback ที่ช่วยให้เราสามารถจัดการ touch event ที่เหมาะสมได้
- เมื่อผู้ใช้สัมผัสหน้าจอ แอปพลิเคชันที่ดี ควรมีคุณสมบัติ
  - Feedback/Highlighting: ควรแสดงให้ผู้ใช้เห็น เมื่อผู้ใช้มีการสัมผัสหรือปล่อยหน้าจอ
  - Cancel-ability: ผู้ใช้สามารถยกเลิกการทำ action ระหว่างที่สัมผัสหน้าจอได้

Responder Lifecycle

- Responder system จะพิจารณา View ที่เป็นเจ้าของ interaction lock ณ เวลาหนึ่งๆ
  - เรียก View นั้นว่า Responder
  - ขณะที่เกิด touch gesture นั้น interaction lock อาจถูกส่งไปให้ Ancestor view ที่เกี่ยวข้องกับ responder นั้นได้

Responder Lifecycle – Touch start

- View สามารถขอใช้ (request) interaction lock ได้ ด้วยการเรียก
  - View.props.onStartShouldSetResponder: (evt) => true : View นี้ต้องการเป็น responder เมื่อเริ่ม touch เลยหรือไม่
  - View.props.onMoveShouldSetResponder: (evt) => true: ถูกเรียกทุกครั้งที่มีการขยับนิ้วขณะที่สัมผัสจอ View นี้ต้องการเป็น responder หรือไม่
  - ถ้าฟังก์ชันเหล่านี้ รีเทิร์น true หมายถึง View นั้นขอใช้ interaction lock
    - หากปัจจุบัน ไม่มี View ไหนเป็นเจ้าของ interaction lock แล้ว View นี้จะเป็น responder โดยอัตโนมัติ
    - ถ้ามี View อื่นเป็นเจ้าของ interaction lock อยู่แล้ว onResponderTerminationRequest ของ View นั้นจะถูกเรียก ซึ่ง View นั้นเลือกได้ว่าจะยังคงเก็บ interaction lock อยู่หรือไม่

Responder Lifecycle – Request Interaction Lock

- เมื่อ View ต้องการเป็น responder จะเกิดการเรียกฟังก์ชันเหล่านี้
  - View.props.onResponderGrant: (evt) => {} : การขอใช้ interaction lock ได้รับอนุญาต และ View นั้นจะกลายเป็น responder และสามารถตอบสนอง touch event ได้ เช่น ทำ highlight เป็นต้น
  - View.props.onResponderReject: (evt) => {} : การขอใช้ interaction lock ถูกปฏิเสธ

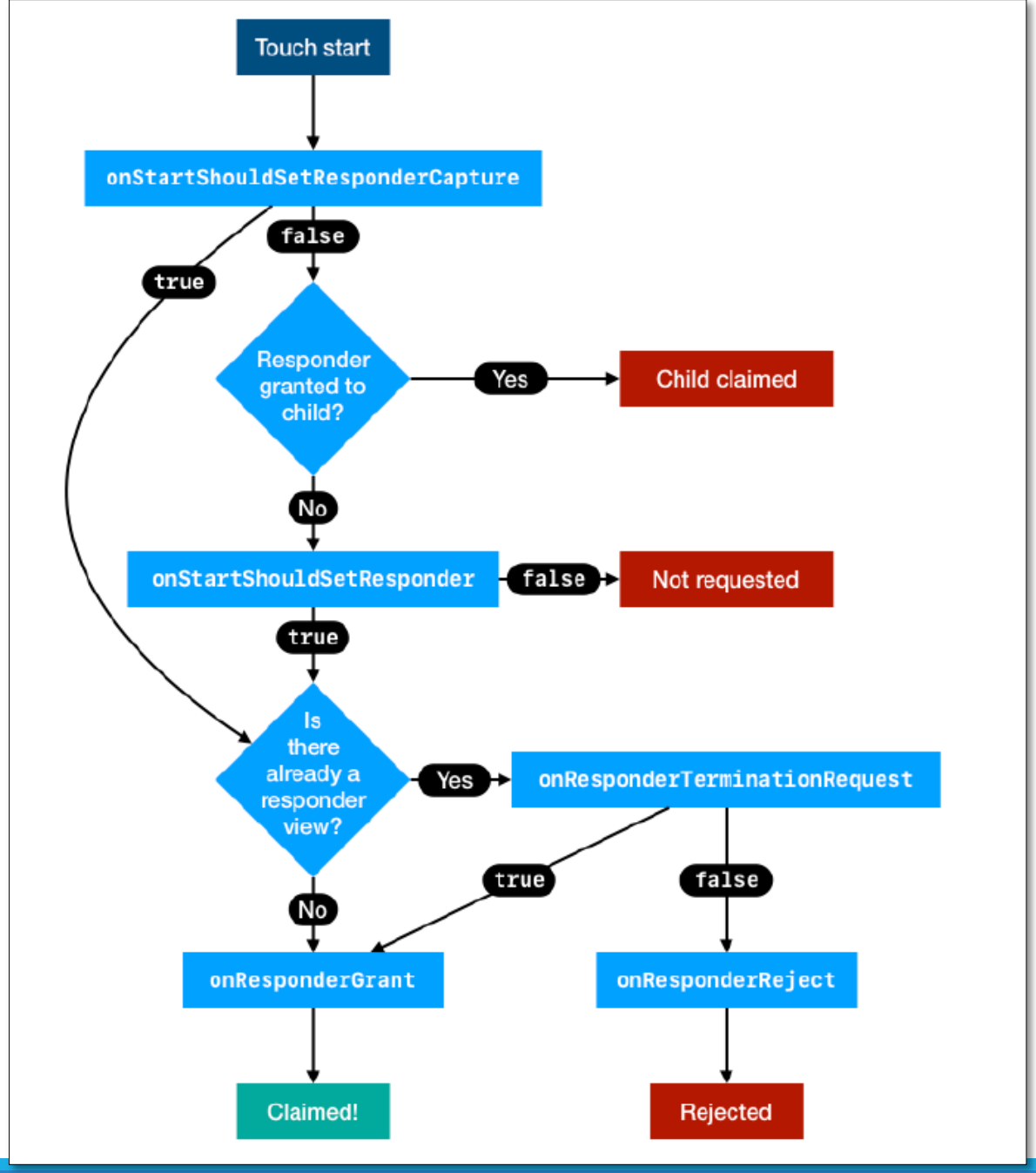
Responder Lifecycle – Responder view

- เมื่อ View ได้เป็น responder แล้ว จะสามารถฟังก์ชันจัดการต่างๆ ดังนี้
  - View.props.onResponderMove: (evt) => {} : จะถูกเรียกเมื่อผู้ใช้ขยับนิ้ว
  - View.props.onResponderRelease: (evt) => {} : จะถูกเรียกเมื่อผู้ใช้ปล่อยนิ้ว
  - View.props.onResponderTerminationRequest: (evt) => true : เมื่อมี View อื่น ขอใช้ interaction lock ซึ่งถ้าไม่อนุญาตให้ View อื่นใช้และยังคงเก็บ lock ไว้ จะรีเทิร์น false มิฉะนั้นจะรีเทิร์น true
  - View.props.onResponderTerminate: (evt) => {} : View นี้ไม่ได้เป็น responder แล้ว (ไม่มี interaction lock) ฟังก์ชันนี้อาจถูกเรียกเมื่อ
    - View อื่นเรียก onResponderTerminationRequest หรือ
    - เมื่อ OS ต้องการใช้ interaction lock

Capture Phase

- onStartShouldSetResponder และ onMoveShouldSetResponder จะถูกเรียกในรูปแบบ bubble handler
  - นั่นคือ คอมโพเนนต์ที่อยู่ลึกที่สุดจะได้เป็น responder กรณีที่มีหลาย View ต้องการเป็น responder
- แต่บางครั้ง parent view ก็ต้องการมั่นใจว่าจะได้เป็น responder ซึ่งสามารถทำได้โดยใช้ Capture handler โดยเรียก
  - View.props.onStartShouldSetResponderCapture: (evt) => true,
  - View.props.onMoveShouldSetResponderCapture: (evt) => true,

Diagram – Responder lifecycle for new touches



Touch Event Object

- Responder function จะถูกเรียกด้วย event object (evt) ซึ่งมี property nativeEvent อันประกอบด้วย
  - changedTouches - Array of all touch events that have changed since the last event
  - identifier - The ID of the touch
  - locationX - The X position of the touch, relative to the responder view
  - locationY - The Y position of the touch, relative to the responder view
  - pageX - The X position of the touch, relative to the root element
  - pageY - The Y position of the touch, relative to the root element
  - target - The node id of the element receiving the touch event
  - timestamp - A time identifier for the touch, useful for velocity calculation
  - touches - Array of all current touches on the screen

PanResponder

- Touch event object มีข้อมูลดิบต่างๆ เช่น เวลาและตำแหน่งการสัมผัส แต่ในบางครั้งแอปพลิเคชันอาจต้องการข้อมูลระยะทางและความเร็วในการสัมผัส
  - React Native ได้เตรียม PanResponder (API ระดับสูง) มาจัดการข้อมูลเหล่านี้
- PanResponder สามารถจัดการ gestureState ซึ่งประกอบด้วยข้อมูลต่างๆ เช่น ระยะทาง และความเร็ว เป็นต้น

PanResponder.create()

- เราสามารถสร้าง PanResponder ด้วย PanResponder.create(config)
- Config object ประกอบด้วยฟังก์ชัน ดังนี้
  - onStartShouldSetPanResponder: (e, gestureState) => {}
  - onStartShouldSetPanResponderCapture: (e, gestureState) => {}
  - onMoveShouldSetPanResponder: (e, gestureState) => {}
  - onMoveShouldSetPanResponderCapture: (e, gestureState) => {}
  - onPanResponderReject: (e, gestureState) => {}
  - onPanResponderGrant: (e, gestureState) => {}
  - onPanResponderStart: (e, gestureState) => {}
  - onPanResponderEnd: (e, gestureState) => {}
  - onPanResponderRelease: (e, gestureState) => {}
  - onPanResponderMove: (e, gestureState) => {}
  - onPanResponderTerminate: (e, gestureState) => {}
  - onPanResponderTerminationRequest: (e, gestureState) => {}
  - onShouldBlockNativeResponder: (e, gestureState) => {}

แต่ละฟังก์ชันนี้จะครอบ responder function อีกที เช่น onPanResponderMove จะครอบ onResponderMove อีกที

gestureState

- stateID - The id of the gestureState – persisted as long as there's at least one touch on screen
- moveX - The latest screen coordinates of the most recently moved touch
- moveY - The latest screen coordinates of the most recently moved touch
- x0 - The screen coordinates at the time the responder was granted
- y0 - The screen coordinates at the time the responder was granted
- dx - Accumulated distance of the gesture since the touch started
- dy - Accumulated distance of the gesture since the touch started
- vx - Current velocity of the gesture
- vy - Current velocity of the gesture
- numberActiveTouches - Number of touches currently on screen

Animation and Gesture

- การทำ Gesture ต่างๆ สามารถ map เข้ากับ animated value ได้ ด้วยการเรียก Animated.event()
- รูปแบบ : event(argMapping, config?)
- ตัวอย่างเช่น การทำ scrolling ในแนวแกนนอน เราจะต้องทำการ map ค่า event.nativeEvent.contentOffset.x เข้ากับค่า scrollX
- onScroll={Animated.event(

```
// scrollX = e.nativeEvent.contentOffset.x
[{ nativeEvent: {
      contentOffset: {
        x: scrollX
} } }] )}
```

Animated.ValueXY

- เป็นค่า 2D ที่ใช้สำหรับทำ 2D animation โดยจะประกอบด้วย Animated.Value 2 ค่า ได้แก่ x และ y
- ตัวอย่างเมธอดที่ใช้สำหรับ Animated.ValueXY
  - setValue(value) : กำหนดค่า Value
  - setOffset(offset) : กำหนดค่า offset
  - flattenOffset() : รวม (Merge) ค่า offset กับ base value และปรับค่า offset เป็น 0
  - extractOffset() : กำหนดค่า offset ให้เป็น base value และปรับค่า base value เป็น 0
  - getLayout() : แปลงค่า {x,y} ให้เป็น {left,top} เพื่อใช้ใน style
  - getTranslateTransform() : แปลงค่า {x,y} ให้เป็นค่าสำหรับ translation transform ได้

- Read more: https://reactnative.dev/docs/animatedvaluexy

ขั้นตอนการใช้ PanResponder และ Animation

- import PanResponder และ Animated มาใช้
  - import {PanResponder, Animated} from ‘react-native’
- สร้าง PanResponder instance ด้วยการเรียก PanResponder.create() และกำหนด Config (panHandler) ต่างๆ ตาม Slide 12
  - เช่น onStartShouldSetPanResponder, onPanResponderMove, onPanResponderRelease เป็นต้น
- กำหนด Animated.ValueXY()
- ปรับปรุงการกำหนดฟังก์ชันใน PanResponder.create() ซึ่งอาจมีการเรียกใช้ค่าจาก Animated.ValueXY()
- เขียนส่วนแสดงผล เช่น กำหนดแท็ก <Animated.View> โดยกำหนด property ให้มี panHandlers ที่กำหนดใน PanResponder ด้วย
  - <Animated.View {panResponder.panHandlers} />

ตัวอย่างโปรแกรม

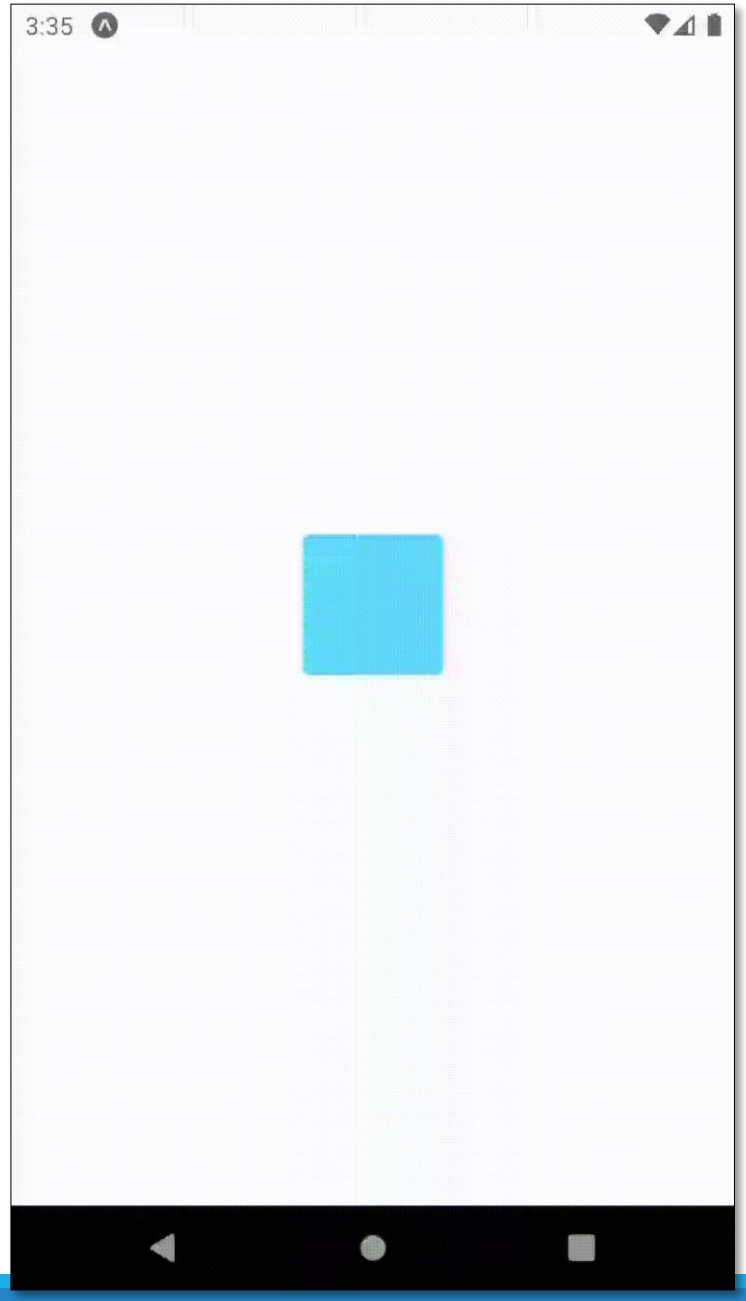
```
import React, { useRef } from "react";
import { Animated, PanResponder, StyleSheet, View } from "react-native";

const Example01 = () => {
 const pan = useRef(new Animated.ValueXY()).current;
 const panResponder = PanResponder.create({
   onStartShouldSetPanResponder: () => true,
   onPanResponderMove: Animated.event([
     null, { dx: pan.x, dy: pan.y, }, ]),
   onPanResponderRelease: () => {
     Animated.spring(
       pan, // Auto-multiplexed
       { toValue: { x: 0, y: 0 }, // Back to zero
         bounciness: 15 }
     ).start(); }, }
 );
```

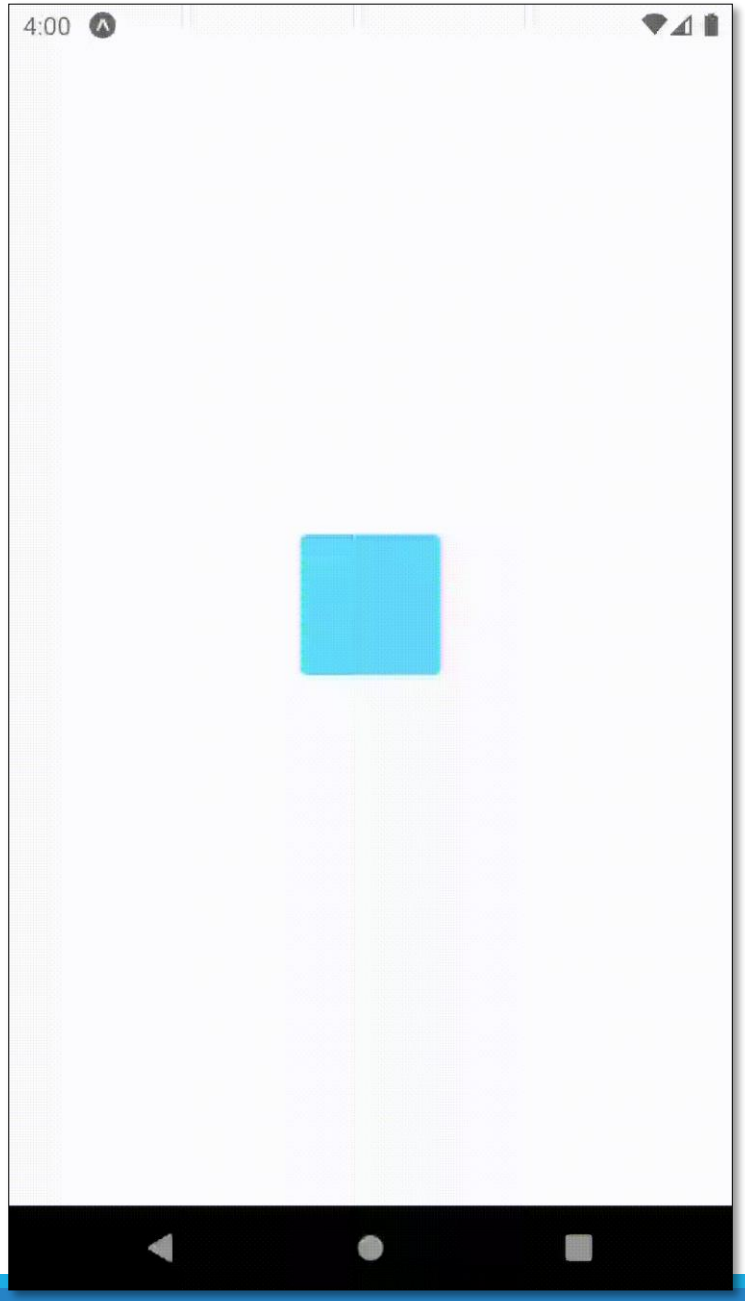
```
return (
  <View style={styles.container}>
    <Animated.View
      {...panResponder.panHandlers}
      style={[pan.getLayout(), styles.box]}
    />
  </View>
);
};
```

ตัวอย่างการทำงานของโปรแกรม

(ซ้าย) – ไม่กำหนด bounciness

(ขวา) – bounciness = 15





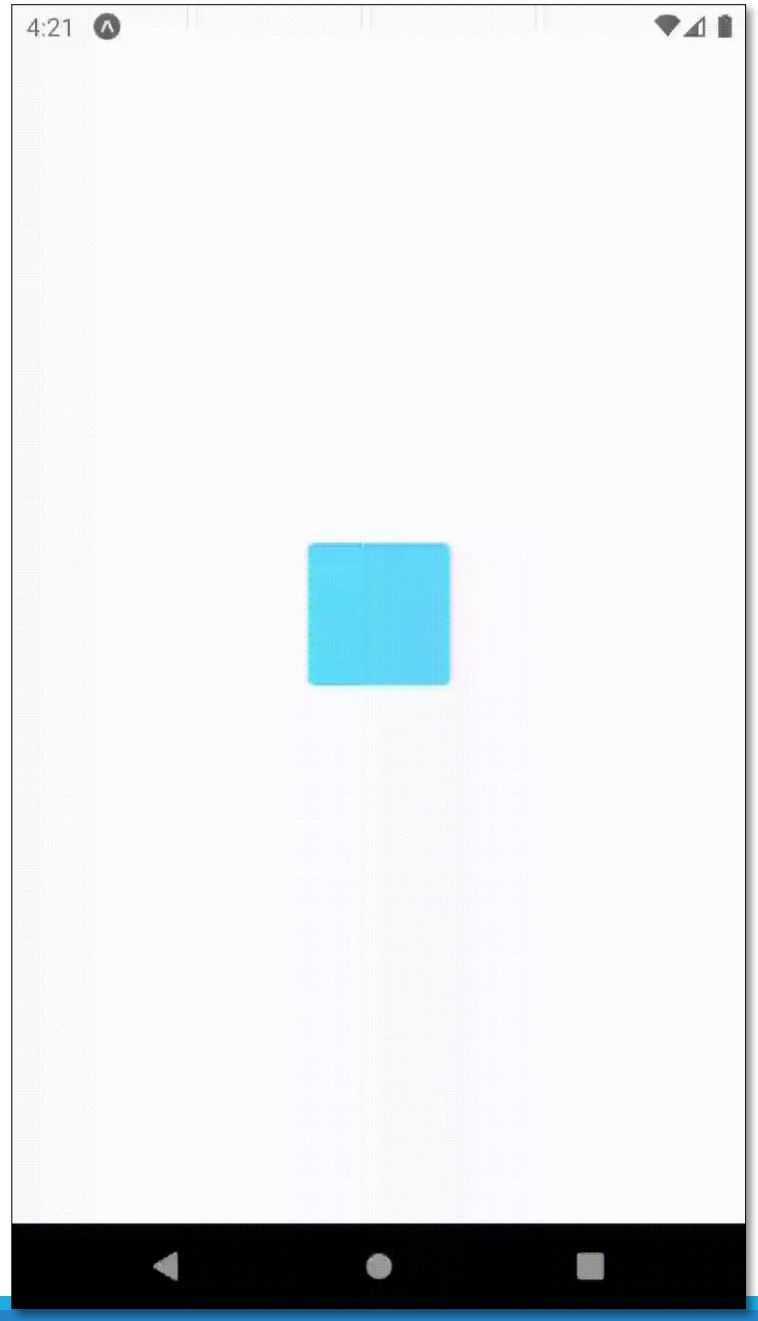
ตัวอย่างโปรแกรม

```
import React, { useRef } from "react";
import { Animated, PanResponder, StyleSheet, View } from "react-native";

const Example02 = () => {
 const pan = useRef(new Animated.ValueXY()).current;
 const panResponder = PanResponder.create({
   onStartShouldSetPanResponder: () => true,
   onPanResponderGrant: () => {
       pan.setOffset({ x: pan.x._value, y: pan.y._value, });
      pan.setValue({ x: 0, y: 0 }); },
   onPanResponderMove: Animated.event([
     null, { dx: pan.x, dy: pan.y, }, ]),
   onPanResponderRelease: () => { pan.flattenOffset(); },
  } );
```

```
return (
  <View style={styles.container}>
    <Animated.View
      {...panResponder.panHandlers}
      style={[pan.getLayout(), styles.box]}
    />
  </View>
 );
};
```

ตัวอย่างการทำงานของโปรแกรม

- เมื่อกดค้างบริเวณกล่องสี่เหลี่ยม แล้วลากไปตามจุดต่างๆ กล่องสี่เหลี่ยมจะเคลื่อนตามนิ้วที่ลากไป
- เมื่อปล่อยนิ้ว กล่องสี่เหลี่ยมจะหยุดเคลื่อนที่ ณ ตำแหน่งล่าสุด ตอนปล่อยนิ้ว



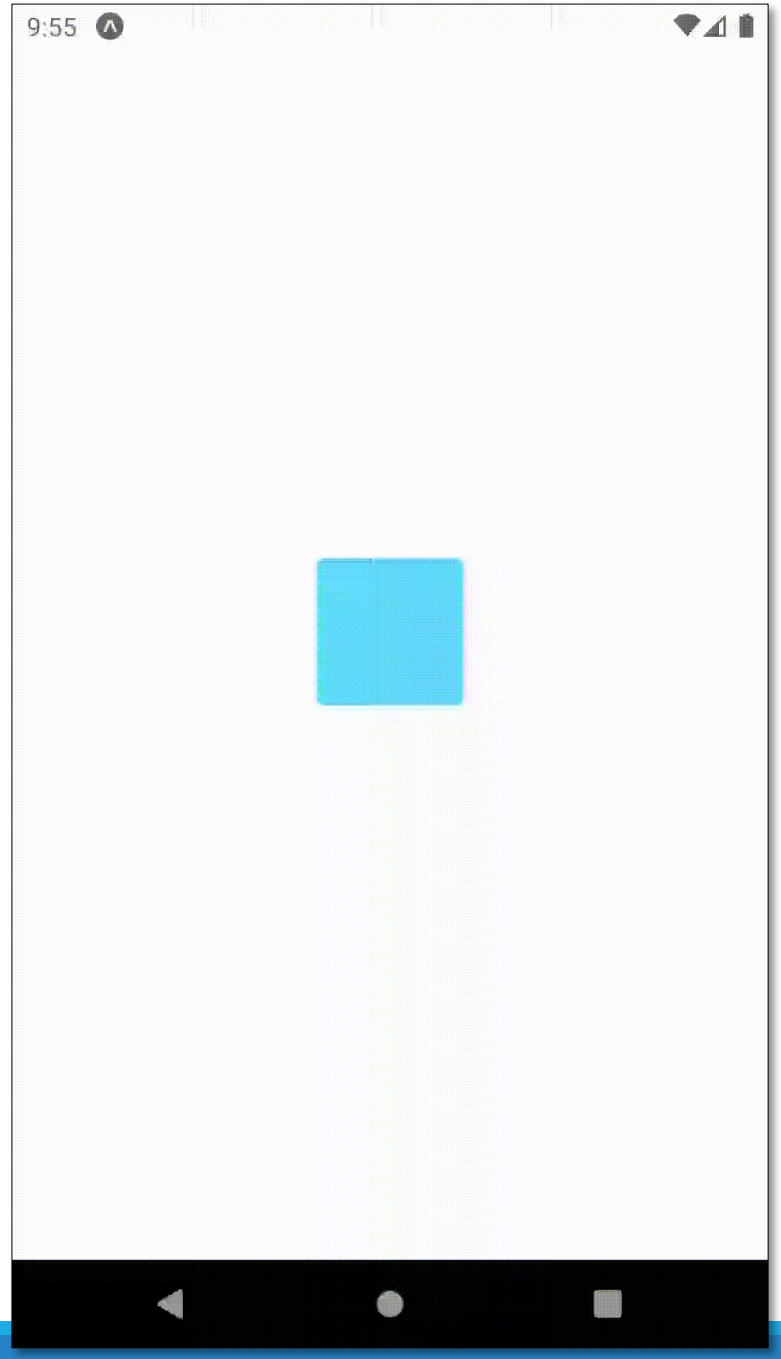
ตัวอย่างโปรแกรม

```
import React, { useRef } from "react";
import { Animated, PanResponder, StyleSheet, View } from "react-native";
const Example03 = () => {
 const pan = useRef(new Animated.ValueXY()).current;
 const scale = useRef(new Animated.Value(1)).current;
 const panResponder = PanResponder.create({
   onStartShouldSetPanResponder: () => true,
   onPanResponderGrant: () => {
       pan.setOffset({ x: pan.x._value, y: pan.y._value, });
       pan.setValue({ x: 0, y: 0 });
       Animated.spring(scale, { toValue:1.5, friction:3 }).start(); },
   onPanResponderMove: Animated.event([
       null, { dx: pan.x, dy: pan.y, }, ]),
   onPanResponderRelease: () => {
       pan.flattenOffset();
       Animated.spring(scale, { toValue:1, friction:3 }).start(); }, });
```

```
return (
  <View style={styles.container}>
    <Animated.View
      {...panResponder.panHandlers}
      style={[ pan.getLayout(),
                styles.box,
                { transform: [{ scale: scale }] }
              ]}
    />
  </View>
 );
};
```

ตัวอย่างการทำงานของโปรแกรม

- คล้ายกับตัวอย่างก่อนหน้า
- เมื่อกดค้างบริเวณกล่องสี่เหลี่ยม กล่องจะมีการขยายขนาดเพิ่มขึ้น และสามารถลากไปตามจุดต่างๆ ได้
- เมื่อปล่อยนิ้ว กล่องสี่เหลี่ยมจะหยุดเคลื่อนที่ และลดขนาดเป็นขนาดเริ่มต้น



ตัวอย่างโปรแกรม

```
import React, { useRef } from "react";
import { Animated, PanResponder, StyleSheet, View } from "react-native";

const Example04 = () => {
  const scale = useRef(new Animated.Value(1)).current;
  const panResponder = PanResponder.create({
    onStartShouldSetPanResponder: () => true,
    onPanResponderMove: (evt, gestureState) => {
      const touches = evt.nativeEvent.touches;
      if (touches.length >= 2) {
        Animated.spring(scale, {
          toValue: 3,
          friction: 3,
          useNativeDriver: false,
        }).start();
      } },
    onPanResponderRelease: () => {
      Animated.spring(scale, {
        toValue: 1,
        friction: 3,
        useNativeDriver: false,
      }).start();  },  });

  return (
    <View style={styles.container}>
      <Animated.View
        {...panResponder.panHandlers}
        style={[styles.box, { transform: [{ scale: scale }] }]}
      />
    </View>
  );
};
```

ตัวอย่างการทำงานของโปรแกรม

- เป็นการสัมผัสหน้าจอแบบ 2 จุดสัมผัส
- เมื่อกดนิ้ว 2 นิ้ว บริเวณกล่องสี่เหลี่ยม แล้วลากนิ้วทั้งสองออกจากกัน จะทำให้กล่องนั้นขยายขนาดใหญ่ขึ้น
- เมื่อปล่อยนิ้ว ขนาดของกล่องจะย่อขนาดเท่ากับตอนเริ่ม