CSE2MAD LAB 9 – GESTURES

AIMS

Movement in Android that has a particular high level context is called a gesture. These include, swiping, scrolling, flinging etc. In this lab you will conduct two exercises,

- Detect and handle an event raised by a particular gesture
- Learn how to track and intuitively analyse the X&Y velocity characteristics for common and custom gestures.

STEP 1 – SET THE CODE FOR THE LAB

```
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.view.GestureDetectorCompat;

import android.os.Bundle;
import android.util.Log;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.VelocityTracker;

public class MainActivity extends AppCompatActivity {

    private VelocityTracker mVelocityTracker = null;
    private static final String DEBUG_TAG = "MOTION";
    private GestureDetectorCompat mDetector;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mDetector = new GestureDetectorCompat(this, new MyGestureListener());
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int index = event.getActionIndex();
        int action = event.getActionMasked();
        int pointerId = event.getPointerId(index);
        this.mDetector.onTouchEvent(event);
        switch (action) {
            case MotionEvent.ACTION_DOWN:
                if (mVelocityTracker == null) {
                    mVelocityTracker = VelocityTracker.obtain();
                } else {
                    mVelocityTracker.clear();
                }
                mVelocityTracker.addMovement(event);
                break;
            case MotionEvent.ACTION_MOVE:
                mVelocityTracker.addMovement(event);
                mVelocityTracker.computeCurrentVelocity(1000);
                Log.d(DEBUG_TAG, "X velocity: " +
                        mVelocityTracker.getXVelocity(pointerId));
                Log.d(DEBUG_TAG, "Y velocity: " +
                        mVelocityTracker.getYVelocity(pointerId));
                break;
```

```
            case MotionEvent.ACTION_UP:
                break;
            case MotionEvent.ACTION_CANCEL:
                mVelocityTracker.recycle();
                mVelocityTracker = null;
                break;
        }
        return true;
    }
    class MyGestureListener extends GestureDetector.SimpleOnGestureListener
    {
        private static final String DEBUG_TAG = "Gestures";
        @Override
        public boolean onDown(MotionEvent event) {
            Log.d(DEBUG_TAG,"onDown: " + event.toString());
            return true;
        }
        @Override
        public boolean onFling(MotionEvent event1, MotionEvent event2,
                               float velocityX, float velocityY) {
            Log.d(DEBUG_TAG, "onFling: " + event1.toString() +
                    event2.toString());
            return true;
        }
    }
}
```

Explanation: Our activity is concerned with gesture and velocity, hence we define class members to enable such functionality. As usual, our onCreate method is used to initialise these members. Note: the listener is specified for the new instance of the GestureDetector. We are interested in only a couple of gestures, therefore instead of implementing the large number of callback methods, we create an inner class that contains overridden methods for onDown and onFling.

Compile and execute the code and experiment with the app. Can you trigger either of these callback by moving your finger on the interface? If so, how do you define a fling etc.?

Look now at the code for outputting the velocity values in the X & Y planes to the debug output. On a movement action the code will trigger. Now perform a swipe gesture as if you were dismissing a notification, perform a two finger scroll, try a fling gesture as before. Copy and paste the debug output for 3 instances of such a gesture simulation. Notice the similarity amongst trials of the same gesture. Notice the unique characteristics the make each gesture category unique.

Think about how we could create our own custom gestures and what kind of capability that may bring to an app.

Please refer to ‘Chapter 29 Implementing Custom Gesture and Pinch Recognition on Android’ from your textbook (Android Studio 4. 0 Development Essentials - Java Edition (2020))