Gesture Recognizer for Java ME

This article provides source and usage examples for the Java ME `GestureRecognizer` class; a class that can recognise gestures: tap, tap'n'hold, drag and circle.

25 Apr 2010

Overview

The `GestureRecognizer` class recognizes the following kinds of gesture:

- *tap* - a brief touch in a single position
- *tap'n'hold* - a prolonged touch in a single position
- *drag* - a swift movement from one side of the screen towards the opposite side
- *circle* - a swift circular motion, from the top of the circle, in either clockwise or anti-clockwise directions

Performance can be tweaked by adjusting the values of `DRAG_THRESHOLD` and `HOLD_THRESHOLD`. `HOLD_THRESHOLD` is simply the amount of time a tap must persist before it becomes a tap'n'hold. `DRAG_THRESHOLD` is used to desensitize the algorithm to small movements during a gesture. Currently, this can cause failure to recognize circular gestures if they are drawn too slowly, or if the circle is very small.

Source code

```
/**
 * Recognize a tap, drag or circular gesture, from a series of (x,y) points.
 * @author Graham Hughes
 */
public class GestureRecognizer {
    // constants returned from getGesture()
    private static final int NONE        = 0;
    public static final int TAP          = 1;
    public static final int TAP_N_HOLD   = 2;
    public static final int DRAG_UP      = 3;
    public static final int DRAG_DOWN    = 4;
    public static final int DRAG_LEFT    = 5;
    public static final int DRAG_RIGHT   = 6;
    public static final int CIRCLE_CLOCK = 7;
    public static final int CIRCLE_ANTI  = 8;
    public static final int UNKNOWN      = 9;

    // any movement smaller than this is ignored (treated as "tap" not "drag")
    private static final int DRAG_THRESHOLD = 4;

    // minimum milliseconds for a "hold"
    private static final int HOLD_THRESHOLD = 500;

    // PDX = positive delta X, NDY = negative delta Y, etc.
    private static final int SEEN_PDX_PDY = 1 << 0;
    private static final int SEEN_NDX_PDY = 1 << 1;
    private static final int SEEN_NDX_NDY = 1 << 2;
    private static final int SEEN_PDX_NDY = 1 << 3;
    private static final int SEEN_PDX     = 1 << 4;
    private static final int SEEN_NDX     = 1 << 5;
    private static final int SEEN_PDY     = 1 << 6;
    private static final int SEEN_NDY     = 1 << 7;

    private static final int SEEN_ALL = SEEN_PDX | SEEN_NDX | SEEN_PDY | SEEN_NDY;

    private long startTime;
    private int startX;
    private int startY;
    private int lastX;
    private int lastY;
    private int minX;
    private int minY;
    private int maxX;
    private int maxY;
    private int seen;
    private int[] sequence = new int[4];
    private int sequenceIndex;
    private int gesture;

    /** Create recognizer, with starting (x,y) */
    public GestureRecognizer(int x, int y) {
        startTime = System.currentTimeMillis();
        startX = lastX = minX = maxX = x;
        startY = lastY = minY = maxY = y;
    }

    /** Add (x,y) position to gesture sequence */
    public void addMotion(int newX, int newY) {
        checkIncomplete();

        int dx = newX - lastX;
        int dy = newY - lastY;

        int seenThisTime = 0;

        if (dx < -DRAG_THRESHOLD) {
            seenThisTime |= SEEN_NDX;
            lastX = newX;
            if (newX < minX) {
                minX = newX;
            }
        } else if (dx > DRAG_THRESHOLD) {
            seenThisTime |= SEEN_PDX;
            lastX = newX;
            if (newX > maxX) {
                maxX = newX;
            }
```


(C) Copyright Nokia 2013. All rights reserved.

```
        }

        if (dy < -DRAG_THRESHOLD) {
            seenThisTime |= SEEN_NDY;
            lastY = newY;
            if (newY < minY) {
                minY = newY;
            }
        } else if (dy > DRAG_THRESHOLD) {
            seenThisTime |= SEEN_PDY;
            lastY = newY;
            if (newY > maxY) {
                maxY = newY;
            }
        }

        switch (seenThisTime) {
        case SEEN_PDX | SEEN_PDY:
            seenThisTime |= SEEN_PDX_PDY;
            addToSequence(SEEN_PDX_PDY);
            break;
        case SEEN_PDX | SEEN_NDY:
            seenThisTime |= SEEN_PDX_NDY;
            if (sequenceIndex > 0) {
                addToSequence(SEEN_PDX_NDY);
            }
            break;
        case SEEN_NDX | SEEN_PDY:
            seenThisTime |= SEEN_NDX_PDY;
            addToSequence(SEEN_NDX_PDY);
            break;
        case SEEN_NDX | SEEN_NDY:
            seenThisTime |= SEEN_NDX_NDY;
            if (sequenceIndex > 0) {
                addToSequence(SEEN_NDX_NDY);
            }
            break;
        }

        if (seenThisTime != 0) {
            seen |= seenThisTime;
        }
    }

    /** End the gesture, with terminating (x,y) */
    public void endGesture(int endX, int endY) {
        addMotion(endX, endY);

        if (seen == 0) {
            if (System.currentTimeMillis() > startTime + HOLD_THRESHOLD) {
                gesture = TAP_N_HOLD;
            } else {
                gesture = TAP;
            }
        } else if ((seen & SEEN_ALL) != SEEN_ALL) {
            int dx = maxX - minX;
            int dy = maxY - minY;

            if ((seen & SEEN_NDY) != 0 && (seen & SEEN_PDY) == 0 && (dy > dx)) {
                gesture = DRAG_UP;
            }

            if ((seen & SEEN_PDY) != 0 && (seen & SEEN_NDY) == 0 && (dy > dx)) {
                gesture = DRAG_DOWN;
            }

            if ((seen & SEEN_NDX) != 0 && (seen & SEEN_PDX) == 0 && (dx > dy)) {
                gesture = DRAG_LEFT;
            }

            if ((seen & SEEN_PDX) != 0 && (seen & SEEN_NDX) == 0 && (dx > dy)) {
                gesture = DRAG_RIGHT;
            }
        } else {
            if (sequenceEquals(new int[] {SEEN_PDX_PDY, SEEN_NDX_PDY, SEEN_NDX_NDY, SEEN_PDX_NDY})) {
                gesture = CIRCLE_CLOCK;
            } else if (sequenceEquals(new int[] {SEEN_NDX_PDY, SEEN_PDX_PDY, SEEN_PDX_NDY, SEEN_NDX_NDY})) {
                gesture = CIRCLE_ANTI;
            }
        }

        if (gesture == NONE) {
            gesture = UNKNOWN;
        }
    }

    /** @return the gesture type */
    public int getGesture() {
        checkComplete();
        return gesture;
    }

    /** @return the "x" position for "tap" gestures */
    public int getTapX() {
        checkTap();
        return startX;
    }

    /** @return the "y" position for "tap" gestures */
    public int getTapY() {
        checkTap();
        return startY;
    }

    /** @return the left-x position for the area of a circular gesture */
    public int getAreaX() {
        checkArea();
        return minX;
    }

    /** @return the top-y position for the area of a circular gesture */
    public int getAreaY() {
        checkArea();
        return minY;
    }

    /** @return the width of the area of a circular gesture */
    public int getAreaWidth() {
        checkArea();
        return maxX - minX;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    }

    /** @return the height of the area of a circular gesture */
    public int getAreaHeight() {
        checkArea();
        return maxY - minY;
    }

    private void checkIncomplete() {
        if (gesture != NONE) {
            throw new IllegalStateException("gesture complete");
        }
    }

    private void checkComplete() {
        if (gesture == NONE) {
            throw new IllegalStateException("gesture not complete");
        }
    }

    private void checkTap() {
        checkComplete();
        if (gesture != TAP && gesture != TAP_N_HOLD) {
            throw new IllegalStateException("gesture not a tap");
        }
    }

    private void checkArea() {
        checkComplete();
        if (gesture != CIRCLE_CLOCK && gesture != CIRCLE_ANTI) {
            throw new IllegalStateException("gesture not an area");
        }
    }

    private boolean sequenceContains(int n) {
        boolean contains = false;
        for (int i = 0; i < sequenceIndex; i++) {
            if (sequence[i] == n) {
                contains = true;
            }
        }
        return contains;
    }

    private void addToSequence(int n) {
        if (!sequenceContains(n) && sequenceIndex < sequence.length) {
            System.out.println(sequenceIndex + ": " + n);
            sequence[sequenceIndex] = n;
            sequenceIndex++;
        }
    }

    private boolean sequenceEquals(int[] a) {
        boolean equals;
        if (sequenceIndex == a.length) {
            equals = true;
            for (int i = 0; i < sequenceIndex; i++) {
                if (sequence[i] != a[i]) {
                    equals = false;
                }
            }
        } else {
            equals = false;
        }
        return equals;
    }
}
```

Usage Example

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

public class Gesture extends MIDlet implements CommandListener {

    private Canvas canvas;

    public void startApp() {
        if (canvas == null) {
            canvas = new TestCanvas();
            canvas.addCommand(new Command("Exit", Command.EXIT, 0));
            canvas.setCommandListener(this);
        }
        Display.getDisplay(this).setCurrent(canvas);
    }

    public void pauseApp() {
        // empty
    }

    public void destroyApp(boolean must) {
        // empty
    }

    public void commandAction(Command c, Displayable d) {
        if (c.getCommandType() == Command.EXIT) {
            notifyDestroyed();
        }
    }

    private static class TestCanvas extends Canvas {
        private GestureRecognizer recognizer;
        private int gesture;
        private boolean show;

        protected void paint(Graphics g) {
            g.setColor(0x400000);
            g.fillRect(0, 0, getWidth(), getHeight());

            if (show) {
                g.setColor(0xffff00);
                String s;
                boolean drawTap = false;
                boolean drawArea = false;
```


(C) Copyright Nokia 2013. All rights reserved.

```
                switch (gesture) {
                case GestureRecognizer.TAP:
                    s = "tap";
                    drawTap = true;
                    break;
                case GestureRecognizer.TAP_N_HOLD:
                    s = "tap'n'hold";
                    drawTap = true;
                    break;
                case GestureRecognizer.DRAG_UP:
                    s = "drag up";
                    break;
                case GestureRecognizer.DRAG_DOWN:
                    s = "drag down";
                    break;
                case GestureRecognizer.DRAG_LEFT:
                    s = "drag left";
                    break;
                case GestureRecognizer.DRAG_RIGHT:
                    s = "drag right";
                    break;
                case GestureRecognizer.CIRCLE_CLOCK:
                    s = "circle clockwise";
                    drawArea = true;
                    break;
                case GestureRecognizer.CIRCLE_ANTI:
                    s = "circle anti-clockwise";
                    drawArea = true;
                    break;
                default:
                    s = "unknown";
                    break;
                }

                g.drawString(s, getWidth() / 2, getHeight(), Graphics.HCENTER | Graphics.BOTTOM);

                if (drawTap) {
                    g.drawRect(recognizer.getTapX() - 4, recognizer.getTapY() - 4, 8, 8);
                }
                if (drawArea) {
                    g.drawRect(recognizer.getAreaX(), recognizer.getAreaY(), recognizer.getAreaWidth(), recognizer.getAreaHeight());
                }
            }
        }

        protected void pointerPressed(int x, int y) {
            recognizer = new GestureRecognizer(x, y);

            show = false;
            repaint();
        }

        protected void pointerDragged(int x, int y) {
            recognizer.addMotion(x, y);
        }

        protected void pointerReleased(int x, int y) {
            recognizer.endGesture(x, y);
            gesture = recognizer.getGesture();

            show = true;
            repaint();
        }
    }
}
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.