A Method for More Intelligent Touch Event Processing

Most Likely Widget

Abstract

Touch devices do not provide users with the ability to specify touch event point (x,y) coordinates with the single-pixel precision of desktop mouse-enabled computers. The result is that touch device users must tolerate frequent activation of unintended UI widgets. This slide deck presents a simple, first-draft method for reducing the frequency of unintended UI widget activation by attempting to infer the *most likely widget* which a user intended to activate rather than naively selecting the widget whose bounding rectangle contains the touch point (x,y) coordinate. Because of the prevalent *Rectangle.contains(Point touchPoint)* based event-to-widget routing method (perhaps unwisely adopted from desktop systems) *missing an intended widget by a single pixel can sometimes invoke dangerous actions on an unintended widget* such as opening a malicious e-mail when a user's intention was to click an adjacent selection checkbox so that the suspicious e-mail could be deleted.

Author: Richard Creamer

Website: http://goo.gl/KxGtxQ

Email: 2to32minus1@gmail.com

On the right is a mockup of a typical smartphone e-mail summary app screen. Currently, touch events anywhere within the green rectangle for that (presumed) table cell will be sent to the checkbox. Touch events anywhere within the blue rectangle will cause the e-mail to be displayed.

Even a slight touch positional error such as a touch event in the first pixel column of the 'E' in Email will invoke an unintended and perhaps dangerous action (for instance, opening a malicious e-mail).

In the app mockup to the right, there is a green plus sign at the centroid for the green checkbox active touch area, and a blue plus sign in the e-mail summary text area. Ideally, the red plus sign would be located at the centroid of the checkbox widget vs. its containing table cell, but this is not the case on at least one smartphone platform.

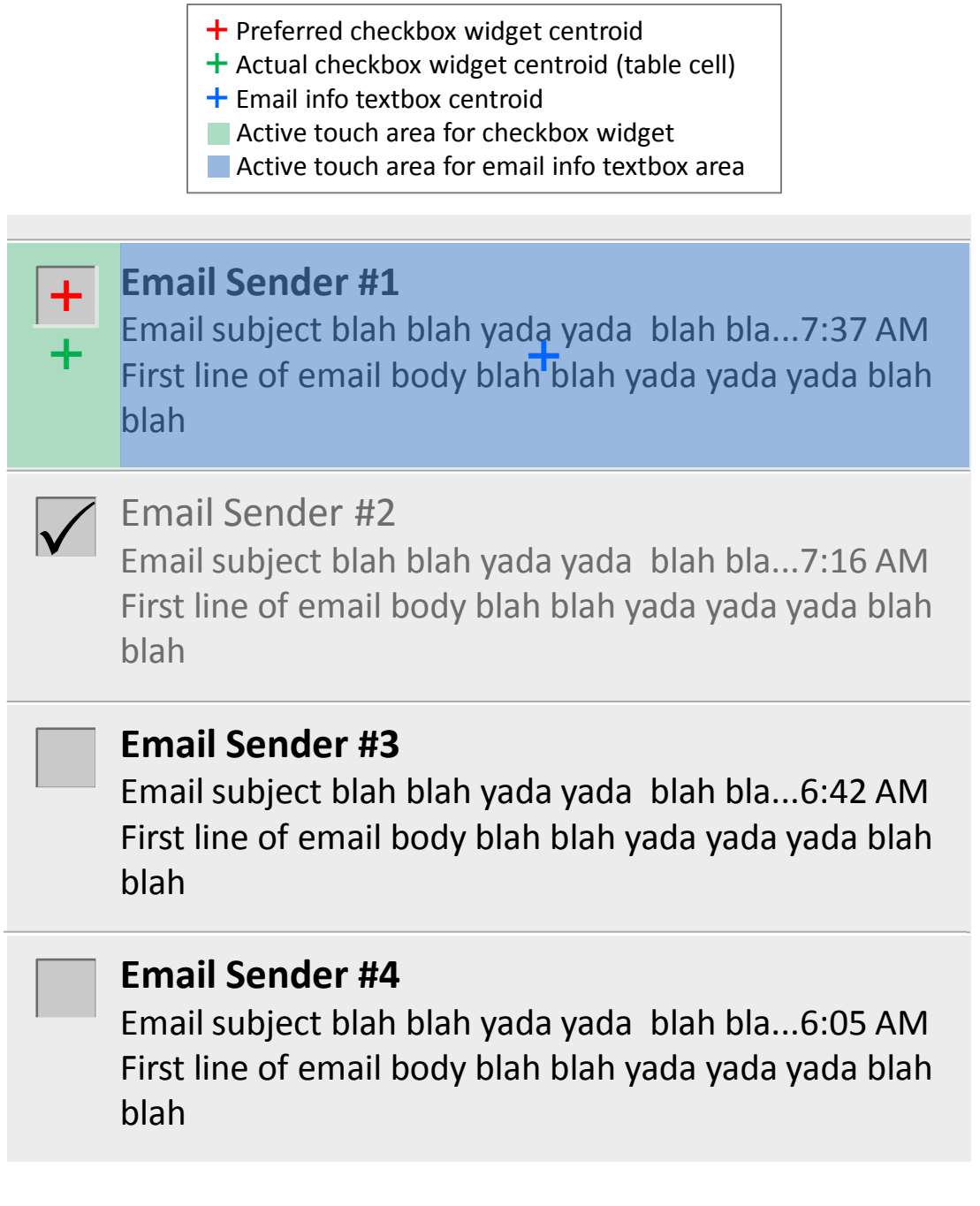
The proposed method is based on:

- Touch point (x,y) to widget centroid distances
- Widget whose bounding rectangle contains (x,y)
- Radius of Confusion: A parameterized constant for the approximate radius of finger touch area/blob in pixels

A simple version of the proposed algorithm:

- For each touch event point (x,y)
  - Determine the widget whose centroid is closest to (x,y)
  - If this distance is less than the Radius of Confusion:
    - Map the event to this widget
  - Else
    - If no widget centroid is within the Radius of Confusion distance:
      - If a widget's bounding rectangle contains (x,y):
        - Map the event to this widget
      - Else
        - Disregard touch event

Note that this is but one simple/illustrative UI context.



Screenshot of Java/Swing prototype implementation exhaustive test with Radius of Confusion = 40 pixels

On the right is a screenshot of an exhaustive test harness implementation of the proposed algorithm.

In the four labeled quadrants:

- **Upper Left**
  - Ideal case where checkbox widget's active area is just the checkbox square (which it is not)
  - Checkbox and e-mail info area centroids are marked with uniquely-colored plus signs
- **Upper Right**
  - Actual case where checkbox widget's container table cell is the active area
  - Checkbox cell and e-mail info area centroids are marked with uniquely-colored plus signs
- **Lower Left and Lower RIght**
  - At each JPanel pixel, a semi-transparent colored pixel was drawn indicating the widget to which a touch event at the pixel's (x,y) coordinates would be mapped w/proposed algorithm
  - Both the lower left and right panels correspond to the panels above them, respectively

As can be seen, the radius of confusion parameter can be used to provide the user with a controllable margin of error in their touch points.

From this test, it appears that a larger Radius of Confusion would perform better.

A simple Java implementation of proposed algorithm from prototype code at end of presentation:

```
////////////////////////////////
// Most Likely Widget Algorithm:  (simple implementation)
//  1) Compute closest widget (with closest centroid) to touch point but which is within radiusOfConfusion
//  2) Compute widget rectangle which actually contains touch point
//  3) Route touch event (x,y) to nearest widget if a nearby widget found, else route to widget w/bounding rect containing (x,y)
// Return null if event should not be handled
////////////////////////////////
private static ColoredRect getMostLikelyWidget( int x, int y, List<ColoredRect> widgets, int radiusOfConfusion ) {
    int minDist = Integer.MAX_VALUE;
    ColoredRect closestWidget = null;
    ColoredRect pointInWidget = null;
    for ( ColoredRect cr : widgets ) {
        Rectangle r = cr.r;
        int xc = r.x + r.width/2;
        int yc = r.y + r.height/2;
        int dist = ( int ) Math.round( Math.sqrt( ( x - xc ) * ( x - xc ) + ( y - yc ) * ( y - yc ) ) );
        if ( dist < minDist && dist < radiusOfConfusion ) { // Step 1
            minDist = dist;
            closestWidget= cr;
        }
        if ( r.contains( x, y ) )  {  // Step 2
            pointInWidget = cr;
        }
    }
    return ( closestWidget != null ) ? closestWidget : pointInWidget;  // Step 3
}

////////////////////////////////
// Helper class
////////////////////////////////
class ColoredRect {
    public final Rectangle r;
    public final Color c;
    public ColoredRect( Rectangle r, Color c ) {
        this.r = r;
        this.c = c;
    }

} // End class: ColoredRect
```

Comments

- The ideas presented herein were inspired by an Android project, Cool Clock, developed by the author at home a few years ago which needed to enable elementary grade children to drag graphical analog clock hands without requiring the touch/drag points to precisely lie within the boundary of each clock hand. *(A similar 'most likely widget' algorithm was developed.)*
- The presented ideas were independently developed. Similarity to other published works is coincidental and unknown to the author.
- A wider range of UI contexts should be fully evaluated if this or a similar algorithm is to be considered for implementation on touch devices for native apps and browser content.
- After evaluating a wider range of UI contexts, additional *'most likely widget'* decision metrics, in addition to centroids, may be necessary.
- It is the opinion of the author that were this or a similar algorithm implemented, a significant reduction in the amount of user frustration with touch devices could be achieved.
- Comments are welcome: 2to32minus1@gmail.com

Simple Java implementation (font size = 5 to fit on single slide - intended for copy/past/test/evaluate)

Link to Main.java: https://sites.google.com/site/rickcreamer/Home/touch-device-most-likely-widget/Main.java

```
//////////////////////
// Copyright (c) 2014 Richard Creamer - All Rights Reserved
//////////////////////

import java.awt.*;
import java.awt.font.LineMetrics;
import java.util.List;
import java.util.ArrayList;
import javax.swing.*;
import javax.swing.border.*;

////////////////////////////
// Main tester class
////////////////////////////

public class Main {

   public static void main( String [] args ) {
      SwingUtilities.invokeLater( new Runnable() {
         @Override
         public void run() {

            // Create e-mail mockup JPanels
            JPanel checkboxCentroidsNoOverlayPanel = makePanelWithInsets( 10, new MockupMailPanel( false, false ) );
            JPanel checkboxCentroidsWithOverlayPanel = makePanelWithInsets( 10, new MockupMailPanel( false, true ) );
            JPanel checkboxCellsNoOverlayPanel = makePanelWithInsets( 10, new MockupMailPanel( true, false ) );
            JPanel checkboxCellsWithOverlayPanel = makePanelWithInsets( 10, new MockupMailPanel( true, true ) );

            // Add titled border inside compound border
            checkboxCentroidsNoOverlayPanel.setBorder( new CompoundBorder( new TitledBorder( " Mail Mockup Checkbox Centroid - No Overlay " ), new EmptyBorder( 0, 10, 10, 10 ) ) );
            checkboxCentroidsWithOverlayPanel.setBorder( new CompoundBorder( new TitledBorder( " Above w/ 'Most Likely Widget' Touch Test Overlay " ), new EmptyBorder( 0, 10, 10, 10 ) ) );
            checkboxCellsNoOverlayPanel.setBorder( new CompoundBorder( new TitledBorder( " Mail Mockup Checkbox Cell Centroid - No Overlay " ), new EmptyBorder( 0, 10, 10, 10 ) ) );
            checkboxCellsWithOverlayPanel.setBorder( new CompoundBorder( new TitledBorder( " Above w/ 'Most Likely Widget' Touch Test Overlay " ), new EmptyBorder( 0, 10, 10, 10 ) ) );

            // Create content pane JPanel
            JPanel holder = new JPanel();
            holder.setBorder( new EmptyBorder( 10, 10, 10, 10 ) );
            GridLayout gl = new GridLayout( 2, 2 );
            gl.setHgap( 5 );
            gl.setVgap( 5 );
            holder.setLayout( gl );
            holder.add( checkboxCentroidsNoOverlayPanel );
            holder.add( checkboxCellsNoOverlayPanel );
            holder.add( checkboxCentroidsWithOverlayPanel );
            holder.add( checkboxCellsWithOverlayPanel );

            // Create JFrame and add content panel, etc...
            JFrame jf = new JFrame();
            jf.setContentPane(  holder );
            jf.setSize( 960, 940 ); // 450, 460 );
            jf.setResizable(  false );
            jf.setTitle( "Most Likely Widget Touch Point Tester" );
            jf.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
            jf.setVisible( true );
         }
      } );
   }

   ////////////////////////////////
   private static JPanel makePanelWithInsets( int insetSize, JPanel content ) {
      JPanel p = new JPanel();
      Border margin = BorderFactory.createEmptyBorder( insetSize, insetSize, insetSize, insetSize );
      p.setBorder( margin );
      p.setLayout( new BorderLayout() );
      p.add( content, BorderLayout.CENTER );
      return p;
   }

} // End class: Main

//////////////////////////////
// Class which draws a mockup of a smartphone e-mail app
//////////////////////////////

class MockupMailPanel extends JPanel {

   // Static data
   private static final int CONTROL_HORIZ_MARGIN = 0;
   private static final int LINE_SEP_HEIGHT = 2;
   private static final int PANEL_CELL_HEIGHT = 95;
   private static final int VERT_MARGIN = 20; // Gap between top of window and beginning of mockup email panel
   private static final Color BKG_CLR = new Color( 236, 236, 236 );
   private static final Color LINE_DARK_CLR = new Color( 166, 166, 166 );
   private static final Color LINE_LITE_CLR = new Color( 255, 255, 255 );
   private static final Color CHECKBOX_LINE_CLR = new Color( 109, 109, 109 );
   private static final Color CHECKBOX_FILL_CLR = new Color( 201, 201, 201 );
   private static final int CHECKBOX_SIZE = 22;
   private static final int CELL_MARGIN = 15;
   private static final int MAIL_TEXT_INSET = 2 * CELL_MARGIN + CHECKBOX_SIZE;
   private static final String [] MAIL_TIMES = { "7:37 AM", "7:16 AM", "6:42 AM", "6:05 AM" };
   private static final int numEmails = MAIL_TIMES.length;
   private static final Color [] checkboxCentroidColors = { Color.RED, new Color( 0x4a, 0xb2, 0x3a ), Color.BLUE, new Color( 0x4a, 0xb2, 0xff ) };
   private static final Color [] emailTextAreaCentroidColors = { Color.CYAN, Color.MAGENTA, new Color( 0x9a, 0x6e, 0x04 ), new Color( 0xff, 0x77, 0x04 ) };
   private static final Font font = new Font( "Arial", Font.PLAIN, 16 );

   // Instance data
   private List<Rectangle> checkboxRects = new ArrayList<>();
   private List<Rectangle> checkboxCellRects = new ArrayList<>();
   private List<Rectangle> mailCellRects = new ArrayList<>();
   private final boolean useCellCentroidsForCheckboxes;
   private final boolean drawOverlay;

   ////////////////////////////////
   public MockupMailPanel( boolean useCellCentroidsForCheckboxes, boolean drawOverlay ) {
      this.useCellCentroidsForCheckboxes = useCellCentroidsForCheckboxes;
      this.drawOverlay = drawOverlay;
   }

   ////////////////////////////////
   @Override
   public void paintComponent( Graphics g ) {
      Graphics2D g2d = ( Graphics2D ) g;
      g2d.setRenderingHint( java.awt.RenderingHints.KEY_ANTIALIASING, java.awt.RenderingHints.VALUE_ANTIALIAS_ON );

      g2d.setColor( BKG_CLR );
      g2d.fillRect( 0, 0, getWidth(), getHeight() );
      int x = 0;
      int y = VERT_MARGIN;
      for ( int i = 0; i < numEmails; ++i ) {
         drawMailEntry( i, g2d, x, y );
         y += PANEL_CELL_HEIGHT + 2; // 2 pixels for the separator lines
      }
      drawCentroids( g2d );
      if ( drawOverlay )
         drawMostLikelyWidgetOverlay( g2d );
   }

   ////////////////////////////////
   private void drawMailEntry( int emailNum, Graphics2D g2d, int x, int y ) {

      // Draw separator lines
      g2d.setColor( LINE_DARK_CLR );
      g2d.drawLine( x, y, getWidth(), y );
      ++y;
      g2d.setColor( LINE_LITE_CLR );
      g2d.drawLine( x, y, getWidth(), y );
      ++y;

      // Draw checkbox and save rectangle for later
      Rectangle r = new Rectangle( x + CELL_MARGIN, y + CELL_MARGIN, CHECKBOX_SIZE, CHECKBOX_SIZE );
      if ( checkboxRects.size() < numEmails )
         checkboxRects.add( r );
      drawCheckBox( g2d, r );

      // Compute and save CELL rectangle in which checkbox is contained
      r = new Rectangle( x, y, MAIL_TEXT_INSET, PANEL_CELL_HEIGHT );
      if ( checkboxCellRects.size() < numEmails )
         checkboxCellRects.add( r );

      // Draw email text summary cell...

      // First compute cell rectangle containing text
      r = new Rectangle( x + MAIL_TEXT_INSET, y, getWidth() - 2 * CELL_MARGIN, PANEL_CELL_HEIGHT );
      if ( mailCellRects.size() < numEmails )
         mailCellRects.add( r );

      // Create mockup e-mail header text lines
      String [] textLines = new String [] { "Email Sender #" + ( emailNum + 1 ),
                                            "Email subject blah blah yada yada blah blah..." + MAIL_TIMES[ emailNum ],
                                            "First line of email body blah blah blah yada yada blah",
                                            "blah" };

      // Draw the text lines for the mockup e-mail
      int textX = x + MAIL_TEXT_INSET;
      int textY = y + CELL_MARGIN;
      g2d.setColor( Color.BLACK );
      g2d.setFont( font );
      FontMetrics fm = getFontMetrics( g2d.getFont() );
      for ( int i = 0; i < textLines.length; ++i ) {
         LineMetrics lm = fm.getLineMetrics( textLines[ i ], g2d );
         g2d.drawString( textLines[ i ], textX, textY + lm.getAscent() - 3 ); // The 3 is a fudge factor for this quickly hacked code
         textY += lm.getHeight();
      }
   }

   ////////////////////////////////
   private void drawCentroids( Graphics2D g2d ) {

      // Draw checkbox centroid crosshairs
      for ( int i = 0; i < numEmails; ++i ) {
         Rectangle r = ( useCellCentroidsForCheckboxes ) ? checkboxCellRects.get( i ) : checkboxRects.get( i );
         drawCrosshairInRect( g2d, r, checkboxCentroidColors[ i ] );
      }

      // Draw email text area cell centroid crosshairs
      for ( int i = 0; i < numEmails; ++i ) {
         Rectangle r = mailCellRects.get( i );
         drawCrosshairInRect( g2d, r, emailTextAreaCentroidColors[ i ] );
      }
   }

   ////////////////////////////////
   private void drawMostLikelyWidgetOverlay( Graphics2D g2d ) {

      // Do not route touch events to widgets farther than this distance
      int radiusOfConfusion = 70;

      // Make single list of special purpose colored rectangle objects to simplify(? ;-) this algorithm eval code
      List<ColoredRect> cr = new ArrayList<>();
      List<Rectangle> cbRects = ( useCellCentroidsForCheckboxes ) ? checkboxCellRects : checkboxRects;
      for ( int i = 0; i < numEmails; ++i ) {
         cr.add( new ColoredRect( cbRects.get( i ), checkboxCentroidColors[ i ] ) );
         cr.add( new ColoredRect( mailCellRects.get( i ), emailTextAreaCentroidColors[ i ] ) );
      }

      // Evaluate every pixel in this JPanel-derived class containing mocked up email summary list
      int w = getWidth();
      int h = getHeight();
      for ( int i = 0; i < h; ++i ) {
         int y = i;
         for ( int j = 0; j < w; ++j ) {
            int x = j;
            ColoredRect mostLikelyWidget = getMostLikelyWidget( x, y, cr, radiusOfConfusion );
            if ( mostLikelyWidget != null )
               drawAlphaPixel( g2d, x, y, mostLikelyWidget.c );
         }
      }
   }

   ////////////////////////////////
   // Most Likely Widget Algorithm:  (simple implementation)
   //  1) Compute closest widget (with closest centroid) to touch point but which is within radiusOfConfusion
   //  2) Compute widget rectangle which actually contains touch point
   //  3) Route touch event (x,y) to nearest widget if a nearby widget found, else route to widget w/bounding rect containing (x,y)
   // Return null if event should not be handled
   ////////////////////////////////
   private static ColoredRect getMostLikelyWidget( int x, int y, List<ColoredRect> widgets, int radiusOfConfusion ) {
      int minDist = Integer.MAX_VALUE;
      ColoredRect closestWidget = null;
      ColoredRect pointInWidget = null;
      for ( ColoredRect cr : widgets ) {
         Rectangle r = cr.r;
         int xc = r.x + r.width/2;
         int yc = r.y + r.height/2;
         int dist = ( int ) Math.round( Math.sqrt( ( x - xc ) * ( x - xc ) + ( y - yc ) * ( y - yc ) ) );
         if ( dist < minDist && dist < radiusOfConfusion ) { // Step 1
            minDist = dist;
            closestWidget= cr;
         }
         if ( r.contains( x, y ) )  {  // Step 2
            pointInWidget = cr;
         }
      }
      return ( closestWidget != null ) ? closestWidget : pointInWidget;  // Step 3
   }

   ////////////////////////////////
   private static void drawAlphaPixel( Graphics2D g2d, int x, int y, Color c ) {
      int overlayAlpha = 50;
      g2d.setColor( new Color( c.getRed(), c.getGreen(), c.getBlue(), overlayAlpha ) );
      g2d.fillRect( x, y, 1, 1 );
   }

   ////////////////////////////////
   private static void drawCheckBox( Graphics2D g2d, Rectangle r ) {
      g2d.setColor( CHECKBOX_FILL_CLR );
      g2d.fillRect( r.x, r.y, r.width, r.height );
      g2d.setColor( CHECKBOX_LINE_CLR );
      g2d.drawLine( r.x, r.y, r.x + CHECKBOX_SIZE - 1, r.y );
      g2d.drawLine( r.x, r.y, r.x, r.y + CHECKBOX_SIZE - 1 );
   }

   ////////////////////////////////
   private static void drawCrosshairInRect( Graphics2D g2d, Rectangle r, Color c ) {
      int crosshairRadius = 7;
      int strokeSize = 3;
      int xc = r.x + r.width/2;
      int yc = r.y + r.height/2;
      g2d.setColor( c );
      g2d.setStroke( new BasicStroke( strokeSize ) );
      g2d.drawLine( xc - crosshairRadius,  yc, xc + crosshairRadius, yc );
      g2d.drawLine( xc, yc - crosshairRadius, xc, yc + crosshairRadius );
   }

} // End class: MockupMailPanel

////////////////////////////////
// Helper class
////////////////////////////////

final class ColoredRect {
   public final Rectangle r;
   public final Color c;
   public ColoredRect( Rectangle r, Color c ) {
      this.r = r;
      this.c = c;
   }

} // End class: ColoredRect
```

A Method for More Intelligent Touch Event Processing

Most Likely Widget

Richard Creamer
Website: http://goo.gl/KxGtxQ
Email: 2to32minus1@gmail.com