|  | /\* |
| --- | --- |
|  | \* AP(r) Computer Science GridWorld Case Study: |
|  | \* Copyright(c) 2002-2006 College Entrance Examination Board |
|  | \* (http://www.collegeboard.com). |
|  | \* |
|  | \* This code is free software; you can redistribute it and/or modify |
|  | \* it under the terms of the GNU General Public License as published by |
|  | \* the Free Software Foundation. |
|  | \* |
|  | \* This code is distributed in the hope that it will be useful, |
|  | \* but WITHOUT ANY WARRANTY; without even the implied warranty of |
|  | \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the |
|  | \* GNU General Public License for more details. |
|  | \* |
|  | \* @author Julie Zelenski |
|  | \* @author Cay Horstmann |
|  | \*/ |
|  |  |
|  | package info.gridworld.gui; |
|  |  |
|  | import info.gridworld.grid.Grid; |
|  | import info.gridworld.grid.Location; |
|  |  |
|  | import java.awt.Color; |
|  | import java.awt.Component; |
|  | import java.awt.Dimension; |
|  | import java.awt.Font; |
|  | import java.awt.Graphics; |
|  | import java.awt.Graphics2D; |
|  | import java.awt.Insets; |
|  | import java.awt.Point; |
|  | import java.awt.Rectangle; |
|  | import java.awt.RenderingHints; |
|  | import java.awt.event.ActionEvent; |
|  | import java.awt.event.ActionListener; |
|  | import java.awt.event.MouseEvent; |
|  | import java.awt.font.FontRenderContext; |
|  | import java.awt.font.LineMetrics; |
|  | import java.awt.geom.Rectangle2D; |
|  | import java.text.MessageFormat; |
|  | import java.util.ArrayList; |
|  | import java.util.ResourceBundle; |
|  |  |
|  | import javax.swing.JPanel; |
|  | import javax.swing.JToolTip; |
|  | import javax.swing.JViewport; |
|  | import javax.swing.Scrollable; |
|  | import javax.swing.SwingConstants; |
|  | import javax.swing.SwingUtilities; |
|  | import javax.swing.Timer; |
|  | import javax.swing.ToolTipManager; |
|  |  |
|  | /\*\* |
|  | \* A <code>GridPanel</code> is a panel containing a graphical display of the |
|  | \* grid occupants. <br /> |
|  | \* This code is not tested on the AP CS A and AB exams. It contains GUI |
|  | \* implementation details that are not intended to be understood by AP CS |
|  | \* students. |
|  | \*/ |
|  |  |
|  | public class GridPanel extends JPanel implements Scrollable, |
|  | PseudoInfiniteViewport.Pannable |
|  | { |
|  | private static final int MIN\_CELL\_SIZE = 12; |
|  | private static final int DEFAULT\_CELL\_SIZE = 48; |
|  | private static final int DEFAULT\_CELL\_COUNT = 10; |
|  | private static final int TIP\_DELAY = 1000; |
|  |  |
|  | private Grid<?> grid; |
|  | private int numRows, numCols, originRow, originCol; |
|  | private int cellSize; // the size of each cell, EXCLUDING the gridlines |
|  | private boolean toolTipsEnabled; |
|  | private Color backgroundColor = Color.WHITE; |
|  | private ResourceBundle resources; |
|  | private DisplayMap displayMap; |
|  | private Location currentLocation; |
|  | private Timer tipTimer; |
|  | private JToolTip tip; |
|  | private JPanel glassPane; |
|  |  |
|  | /\*\* |
|  | \* Construct a new GridPanel object with no grid. The view will be |
|  | \* empty. |
|  | \*/ |
|  | public GridPanel(DisplayMap map, ResourceBundle res) |
|  | { |
|  | displayMap = map; |
|  | resources = res; |
|  | setToolTipsEnabled(true); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Paint this component. |
|  | \* @param g the Graphics object to use to render this component |
|  | \*/ |
|  | public void paintComponent(Graphics g) |
|  | { |
|  | Graphics2D g2 = (Graphics2D) g; |
|  |  |
|  | super.paintComponent(g2); |
|  | if (grid == null) |
|  | return; |
|  |  |
|  | Insets insets = getInsets(); |
|  | g2.setColor(backgroundColor); |
|  | g2.fillRect(insets.left, insets.top, numCols \* (cellSize + 1) + 1, numRows |
|  | \* (cellSize + 1) + 1); |
|  |  |
|  | drawWatermark(g2); |
|  | drawGridlines(g2); |
|  | drawOccupants(g2); |
|  | drawCurrentLocation(g2); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Draw one occupant object. First verify that the object is actually |
|  | \* visible before any drawing, set up the clip appropriately and use the |
|  | \* DisplayMap to determine which object to call upon to render this |
|  | \* particular Locatable. Note that we save and restore the graphics |
|  | \* transform to restore back to normalcy no matter what the renderer did to |
|  | \* to the coordinate system. |
|  | \* @param g2 the Graphics2D object to use to render |
|  | \* @param xleft the leftmost pixel of the rectangle |
|  | \* @param ytop the topmost pixel of the rectangle |
|  | \* @param obj the Locatable object to draw |
|  | \*/ |
|  | private void drawOccupant(Graphics2D g2, int xleft, int ytop, Object obj) |
|  | { |
|  | Rectangle cellToDraw = new Rectangle(xleft, ytop, cellSize, cellSize); |
|  |  |
|  | // Only draw if the object is visible within the current clipping |
|  | // region. |
|  | if (cellToDraw.intersects(g2.getClip().getBounds())) |
|  | { |
|  | Graphics2D g2copy = (Graphics2D) g2.create(); |
|  | g2copy.clip(cellToDraw); |
|  | // Get the drawing object to display this occupant. |
|  | Display displayObj = displayMap.findDisplayFor(obj.getClass()); |
|  | displayObj.draw(obj, this, g2copy, cellToDraw); |
|  | g2copy.dispose(); |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Draw the gridlines for the grid. We only draw the portion of the |
|  | \* lines that intersect the current clipping bounds. |
|  | \* @param g2 the Graphics2 object to use to render |
|  | \*/ |
|  | private void drawGridlines(Graphics2D g2) |
|  | { |
|  | Rectangle curClip = g2.getClip().getBounds(); |
|  | int top = getInsets().top, left = getInsets().left; |
|  |  |
|  | int miny = Math.max(0, (curClip.y - top) / (cellSize + 1)) \* (cellSize + 1) + top; |
|  | int minx = Math.max(0, (curClip.x - left) / (cellSize + 1)) \* (cellSize + 1) + left; |
|  | int maxy = Math.min(numRows, |
|  | (curClip.y + curClip.height - top + cellSize) / (cellSize + 1)) |
|  | \* (cellSize + 1) + top; |
|  | int maxx = Math.min(numCols, |
|  | (curClip.x + curClip.width - left + cellSize) / (cellSize + 1)) |
|  | \* (cellSize + 1) + left; |
|  |  |
|  | g2.setColor(Color.GRAY); |
|  | for (int y = miny; y <= maxy; y += cellSize + 1) |
|  | for (int x = minx; x <= maxx; x += cellSize + 1) |
|  | { |
|  | Location loc = locationForPoint( |
|  | new Point(x + cellSize / 2, y + cellSize / 2)); |
|  | if (loc != null && !grid.isValid(loc)) |
|  | g2.fillRect(x + 1, y + 1, cellSize, cellSize); |
|  | } |
|  |  |
|  | g2.setColor(Color.BLACK); |
|  | for (int y = miny; y <= maxy; y += cellSize + 1) |
|  | // draw horizontal lines |
|  | g2.drawLine(minx, y, maxx, y); |
|  |  |
|  | for (int x = minx; x <= maxx; x += cellSize + 1) |
|  | // draw vertical lines |
|  | g2.drawLine(x, miny, x, maxy); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Draws the occupants of the grid. |
|  | \* @param g2 the graphics context |
|  | \*/ |
|  | private void drawOccupants(Graphics2D g2) |
|  | { |
|  | ArrayList<Location> occupantLocs = grid.getOccupiedLocations(); |
|  | for (int index = 0; index < occupantLocs.size(); index++) |
|  | { |
|  | Location loc = (Location) occupantLocs.get(index); |
|  |  |
|  | int xleft = colToXCoord(loc.getCol()); |
|  | int ytop = rowToYCoord(loc.getRow()); |
|  | drawOccupant(g2, xleft, ytop, grid.get(loc)); |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Draws a square that marks the current location. |
|  | \* @param g2 the graphics context |
|  | \*/ |
|  | private void drawCurrentLocation(Graphics2D g2) |
|  | { |
|  | if ("hide".equals(System.getProperty("info.gridworld.gui.selection"))) |
|  | return; |
|  | if (currentLocation != null) |
|  | { |
|  | Point p = pointForLocation(currentLocation); |
|  | g2.drawRect(p.x - cellSize / 2 - 2, p.y - cellSize / 2 - 2, |
|  | cellSize + 3, cellSize + 3); |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Draws a watermark that shows the version number if it is < 1.0 |
|  | \* @param g2 the graphics context |
|  | \*/ |
|  | private void drawWatermark(Graphics2D g2) |
|  | { |
|  | if ("hide".equals(System.getProperty("info.gridworld.gui.watermark"))) |
|  | return; |
|  | g2 = (Graphics2D) g2.create(); |
|  | g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, |
|  | RenderingHints.VALUE\_ANTIALIAS\_ON); |
|  | Rectangle rect = getBounds(); |
|  | g2.setPaint(new Color(0xE3, 0xD3, 0xD3)); |
|  | final int WATERMARK\_FONT\_SIZE = 100; |
|  | String s = resources.getString("version.id"); |
|  | if ("1.0".compareTo(s) <= 0) return; |
|  | g2.setFont(new Font("SansSerif", Font.BOLD, WATERMARK\_FONT\_SIZE)); |
|  | FontRenderContext frc = g2.getFontRenderContext(); |
|  | Rectangle2D bounds = g2.getFont().getStringBounds(s, frc); |
|  | float centerX = rect.x + rect.width / 2; |
|  | float centerY = rect.y + rect.height / 2; |
|  | float leftX = centerX - (float) bounds.getWidth() / 2; |
|  | LineMetrics lm = g2.getFont().getLineMetrics(s, frc); |
|  | float baselineY = centerY - lm.getHeight() / 2 + lm.getAscent(); |
|  | g2.drawString(s, leftX, baselineY); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Enables/disables showing of tooltip giving information about the |
|  | \* occupant beneath the mouse. |
|  | \* @param flag true/false to enable/disable tool tips |
|  | \*/ |
|  | public void setToolTipsEnabled(boolean flag) |
|  | { |
|  | if ("hide".equals(System.getProperty("info.gridworld.gui.tooltips"))) |
|  | flag = false; |
|  | if (flag) |
|  | ToolTipManager.sharedInstance().registerComponent(this); |
|  | else |
|  | ToolTipManager.sharedInstance().unregisterComponent(this); |
|  | toolTipsEnabled = flag; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Sets the grid being displayed. Reset the cellSize to be the |
|  | \* largest that fits the entire grid in the current visible area (use |
|  | \* default if grid is too large). |
|  | \* @param gr the grid to display |
|  | \*/ |
|  | public void setGrid(Grid<?> gr) |
|  | { |
|  | currentLocation = new Location(0, 0); |
|  | JViewport vp = getEnclosingViewport(); // before changing, reset |
|  | // scroll/pan position |
|  | if (vp != null) |
|  | vp.setViewPosition(new Point(0, 0)); |
|  |  |
|  | grid = gr; |
|  | originRow = originCol = 0; |
|  |  |
|  | if (grid.getNumRows() == -1 && grid.getNumCols() == -1) |
|  | { |
|  | numRows = numCols = 2000; |
|  | // This determines the "virtual" size of the pan world |
|  | } |
|  | else |
|  | { |
|  | numRows = grid.getNumRows(); |
|  | numCols = grid.getNumCols(); |
|  | } |
|  | recalculateCellSize(MIN\_CELL\_SIZE); |
|  | } |
|  |  |
|  | // private helpers to calculate extra width/height needs for borders/insets. |
|  | private int extraWidth() |
|  | { |
|  | return getInsets().left + getInsets().right; |
|  | } |
|  |  |
|  | private int extraHeight() |
|  | { |
|  | return getInsets().top + getInsets().left; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Returns the desired size of the display, for use by layout manager. |
|  | \* @return preferred size |
|  | \*/ |
|  | public Dimension getPreferredSize() |
|  | { |
|  | return new Dimension(numCols \* (cellSize + 1) + 1 + extraWidth(), |
|  | numRows \* (cellSize + 1) + 1 + extraHeight()); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Returns the minimum size of the display, for use by layout manager. |
|  | \* @return minimum size |
|  | \*/ |
|  | public Dimension getMinimumSize() |
|  | { |
|  | return new Dimension(numCols \* (MIN\_CELL\_SIZE + 1) + 1 + extraWidth(), |
|  | numRows \* (MIN\_CELL\_SIZE + 1) + 1 + extraHeight()); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Zooms in the display by doubling the current cell size. |
|  | \*/ |
|  | public void zoomIn() |
|  | { |
|  | cellSize \*= 2; |
|  | revalidate(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Zooms out the display by halving the current cell size. |
|  | \*/ |
|  | public void zoomOut() |
|  | { |
|  | cellSize = Math.max(cellSize / 2, MIN\_CELL\_SIZE); |
|  | revalidate(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Pans the display back to the origin, so that 0, 0 is at the the upper |
|  | \* left of the visible viewport. |
|  | \*/ |
|  | public void recenter(Location loc) |
|  | { |
|  | originRow = loc.getRow(); |
|  | originCol = loc.getCol(); |
|  | repaint(); |
|  | JViewport vp = getEnclosingViewport(); |
|  | if (vp != null) |
|  | { |
|  | if (!isPannableUnbounded() |
|  | || !(vp instanceof PseudoInfiniteViewport)) |
|  | vp.setViewPosition(pointForLocation(loc)); |
|  | else |
|  | showPanTip(); |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Given a Point determine which grid location (if any) is under the |
|  | \* mouse. This method is used by the GUI when creating Fish by clicking on |
|  | \* cells in the display. |
|  | \* @param p the Point in question (in display's coordinate system) |
|  | \* @return the Location beneath the event (which may not be a |
|  | \* valid location in the grid) |
|  | \*/ |
|  | public Location locationForPoint(Point p) |
|  | { |
|  | return new Location(yCoordToRow(p.y), xCoordToCol(p.x)); |
|  | } |
|  |  |
|  | public Point pointForLocation(Location loc) |
|  | { |
|  | return new Point(colToXCoord(loc.getCol()) + cellSize / 2, |
|  | rowToYCoord(loc.getRow()) + cellSize / 2); |
|  | } |
|  |  |
|  | // private helpers to convert between (x,y) and (row,col) |
|  | private int xCoordToCol(int xCoord) |
|  | { |
|  | return (xCoord - 1 - getInsets().left) / (cellSize + 1) + originCol; |
|  | } |
|  |  |
|  | private int yCoordToRow(int yCoord) |
|  | { |
|  | return (yCoord - 1 - getInsets().top) / (cellSize + 1) + originRow; |
|  | } |
|  |  |
|  | private int colToXCoord(int col) |
|  | { |
|  | return (col - originCol) \* (cellSize + 1) + 1 + getInsets().left; |
|  | } |
|  |  |
|  | private int rowToYCoord(int row) |
|  | { |
|  | return (row - originRow) \* (cellSize + 1) + 1 + getInsets().top; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Given a MouseEvent, determine what text to place in the floating tool tip |
|  | \* when the the mouse hovers over this location. If the mouse is over a |
|  | \* valid grid cell. we provide some information about the cell and |
|  | \* its contents. This method is automatically called on mouse-moved events |
|  | \* since we register for tool tips. |
|  | \* @param evt the MouseEvent in question |
|  | \* @return the tool tip string for this location |
|  | \*/ |
|  | public String getToolTipText(MouseEvent evt) |
|  | { |
|  | Location loc = locationForPoint(evt.getPoint()); |
|  | return getToolTipText(loc); |
|  | } |
|  |  |
|  | private String getToolTipText(Location loc) |
|  | { |
|  | if (!toolTipsEnabled || loc == null || !grid.isValid(loc)) |
|  | return null; |
|  | Object f = grid.get(loc); |
|  | if (f != null) |
|  | return MessageFormat.format(resources |
|  | .getString("cell.tooltip.nonempty"), new Object[] |
|  | { loc, f }); |
|  | else |
|  | return MessageFormat.format(resources |
|  | .getString("cell.tooltip.empty"), new Object[] |
|  | { loc, f }); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Sets the current location. |
|  | \* @param loc the new location |
|  | \*/ |
|  | public void setCurrentLocation(Location loc) |
|  | { |
|  | currentLocation = loc; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Gets the current location. |
|  | \* @return the currently selected location (marked with a bold square) |
|  | \*/ |
|  | public Location getCurrentLocation() |
|  | { |
|  | return currentLocation; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Moves the current location by a given amount. |
|  | \* @param dr the number of rows by which to move the location |
|  | \* @param dc the number of columns by which to move the location |
|  | \*/ |
|  | public void moveLocation(int dr, int dc) |
|  | { |
|  | Location newLocation = new Location(currentLocation.getRow() + dr, |
|  | currentLocation.getCol() + dc); |
|  | if (!grid.isValid(newLocation)) |
|  | return; |
|  |  |
|  | currentLocation = newLocation; |
|  |  |
|  | JViewport viewPort = getEnclosingViewport(); |
|  | if (isPannableUnbounded()) |
|  | { |
|  | if (originRow > currentLocation.getRow()) |
|  | originRow = currentLocation.getRow(); |
|  | if (originCol > currentLocation.getCol()) |
|  | originCol = currentLocation.getCol(); |
|  | Dimension dim = viewPort.getSize(); |
|  | int rows = dim.height / (cellSize + 1); |
|  | int cols = dim.width / (cellSize + 1); |
|  | if (originRow + rows - 1 < currentLocation.getRow()) |
|  | originRow = currentLocation.getRow() - rows + 1; |
|  | if (originCol + rows - 1 < currentLocation.getCol()) |
|  | originCol = currentLocation.getCol() - cols + 1; |
|  | } |
|  | else if (viewPort != null) |
|  | { |
|  | int dx = 0; |
|  | int dy = 0; |
|  | Point p = pointForLocation(currentLocation); |
|  | Rectangle locRect = new Rectangle(p.x - cellSize / 2, p.y |
|  | - cellSize / 2, cellSize + 1, cellSize + 1); |
|  |  |
|  | Rectangle viewRect = viewPort.getViewRect(); |
|  | if (!viewRect.contains(locRect)) |
|  | { |
|  | while (locRect.x < viewRect.x + dx) |
|  | dx -= cellSize + 1; |
|  | while (locRect.y < viewRect.y + dy) |
|  | dy -= cellSize + 1; |
|  | while (locRect.getMaxX() > viewRect.getMaxX() + dx) |
|  | dx += cellSize + 1; |
|  | while (locRect.getMaxY() > viewRect.getMaxY() + dy) |
|  | dy += cellSize + 1; |
|  |  |
|  | Point pt = viewPort.getViewPosition(); |
|  | pt.x += dx; |
|  | pt.y += dy; |
|  | viewPort.setViewPosition(pt); |
|  | } |
|  | } |
|  | repaint(); |
|  | showTip(getToolTipText(currentLocation), |
|  | pointForLocation(currentLocation)); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Show a tool tip. |
|  | \* @param tipText the tool tip text |
|  | \* @param pt the pixel position over which to show the tip |
|  | \*/ |
|  | public void showTip(String tipText, Point pt) |
|  | { |
|  | if (getRootPane() == null) |
|  | return; |
|  | // draw in glass pane to appear on top of other components |
|  | if (glassPane == null) |
|  | { |
|  | getRootPane().setGlassPane(glassPane = new JPanel()); |
|  | glassPane.setOpaque(false); |
|  | glassPane.setLayout(null); // will control layout manually |
|  | glassPane.add(tip = new JToolTip()); |
|  | tipTimer = new Timer(TIP\_DELAY, new ActionListener() |
|  | { |
|  | public void actionPerformed(ActionEvent evt) |
|  | { |
|  | glassPane.setVisible(false); |
|  | } |
|  | }); |
|  | tipTimer.setRepeats(false); |
|  | } |
|  | if (tipText == null) |
|  | return; |
|  |  |
|  | // set tip text to identify current origin of pannable view |
|  | tip.setTipText(tipText); |
|  |  |
|  | // position tip to appear at upper left corner of viewport |
|  | tip.setLocation(SwingUtilities.convertPoint(this, pt, glassPane)); |
|  | tip.setSize(tip.getPreferredSize()); |
|  |  |
|  | // show glass pane (it contains tip) |
|  | glassPane.setVisible(true); |
|  | glassPane.repaint(); |
|  |  |
|  | // this timer will hide the glass pane after a short delay |
|  | tipTimer.restart(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Calculate the cell size to use given the current viewable region and the |
|  | \* the number of rows and columns in the grid. We use the largest |
|  | \* cellSize that will fit in the viewable region, bounded to be at least the |
|  | \* parameter minSize. |
|  | \*/ |
|  | private void recalculateCellSize(int minSize) |
|  | { |
|  | if (numRows == 0 || numCols == 0) |
|  | { |
|  | cellSize = 0; |
|  | } |
|  | else |
|  | { |
|  | JViewport vp = getEnclosingViewport(); |
|  | Dimension viewableSize = (vp != null) ? vp.getSize() : getSize(); |
|  | int desiredCellSize = Math.min( |
|  | (viewableSize.height - extraHeight()) / numRows, |
|  | (viewableSize.width - extraWidth()) / numCols) - 1; |
|  | // now we want to approximate this with |
|  | // DEFAULT\_CELL\_SIZE \* Math.pow(2, k) |
|  | cellSize = DEFAULT\_CELL\_SIZE; |
|  | if (cellSize <= desiredCellSize) |
|  | while (2 \* cellSize <= desiredCellSize) |
|  | cellSize \*= 2; |
|  | else |
|  | while (cellSize / 2 >= Math.max(desiredCellSize, MIN\_CELL\_SIZE)) |
|  | cellSize /= 2; |
|  | } |
|  | revalidate(); |
|  | } |
|  |  |
|  | // helper to get our parent viewport, if we are in one. |
|  | private JViewport getEnclosingViewport() |
|  | { |
|  | Component parent = getParent(); |
|  | return (parent instanceof JViewport) ? (JViewport) parent : null; |
|  | } |
|  |  |
|  | // GridPanel implements the Scrollable interface to get nicer behavior in a |
|  | // JScrollPane. The 5 methods below are the methods in that interface |
|  |  |
|  | public int getScrollableUnitIncrement(Rectangle visibleRect, |
|  | int orientation, int direction) |
|  | { |
|  | return cellSize + 1; |
|  | } |
|  |  |
|  | public int getScrollableBlockIncrement(Rectangle visibleRect, |
|  | int orientation, int direction) |
|  | { |
|  | if (orientation == SwingConstants.VERTICAL) |
|  | return (int) (visibleRect.height \* .9); |
|  | else |
|  | return (int) (visibleRect.width \* .9); |
|  | } |
|  |  |
|  | public boolean getScrollableTracksViewportWidth() |
|  | { |
|  | return false; |
|  | } |
|  |  |
|  | public boolean getScrollableTracksViewportHeight() |
|  | { |
|  | return false; |
|  | } |
|  |  |
|  | public Dimension getPreferredScrollableViewportSize() |
|  | { |
|  | return new Dimension(DEFAULT\_CELL\_COUNT \* (DEFAULT\_CELL\_SIZE + 1) + 1 + extraWidth(), |
|  | DEFAULT\_CELL\_COUNT \* (DEFAULT\_CELL\_SIZE + 1) + 1 + extraHeight()); |
|  | } |
|  |  |
|  | // GridPanel implements the PseudoInfiniteViewport.Pannable interface to |
|  | // play nicely with the pan behavior for unbounded view. |
|  | // The 3 methods below are the methods in that interface. |
|  |  |
|  | public void panBy(int hDelta, int vDelta) |
|  | { |
|  | originCol += hDelta / (cellSize + 1); |
|  | originRow += vDelta / (cellSize + 1); |
|  | repaint(); |
|  | } |
|  |  |
|  | public boolean isPannableUnbounded() |
|  | { |
|  | return grid != null && (grid.getNumRows() == -1 || grid.getNumCols() == -1); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Shows a tool tip over the upper left corner of the viewport with the |
|  | \* contents of the pannable view's pannable tip text (typically a string |
|  | \* identifiying the corner point). Tip is removed after a short delay. |
|  | \*/ |
|  | public void showPanTip() |
|  | { |
|  | String tipText = null; |
|  | Point upperLeft = new Point(0, 0); |
|  | JViewport vp = getEnclosingViewport(); |
|  | if (!isPannableUnbounded() && vp != null) |
|  | upperLeft = vp.getViewPosition(); |
|  | Location loc = locationForPoint(upperLeft); |
|  | if (loc != null) |
|  | tipText = getToolTipText(loc); |
|  |  |
|  | showTip(tipText, getLocation()); |
|  | } |
|  | } |