import java.awt.\*;

import javax.swing.\*;

import javax.swing.border.\*;

/\*\*

\* This class is a custom Swing component for representing a single map cell in

\* a 2D map. The cell has several different kinds of state, but the most basic

\* state is whether the cell is passable or not.

\*/

public class JMapCell extends JComponent

{

private static final Dimension CELL\_SIZE = new Dimension(12, 12);

/\*\* True indicates that the cell is an endpoint, either start or finish. \*\*/

boolean endpoint = false;

/\*\* True indicates that the cell is passable; false means it is not. \*\*/

boolean passable = true;

/\*\*

\* True indicates that this cell is part of the path between start and end.

\*\*/

boolean path = false;

/\*\*

\* Construct a new map cell with the specified "passability." An input of

\* true means the cell is passable.

\*\*/

public JMapCell(boolean pass)

{

// Set the preferred cell size, to drive the initial window size.

setPreferredSize(CELL\_SIZE);

setPassable(pass);

}

/\*\* Construct a new map cell, which is passable by default. \*\*/

public JMapCell()

{

// Call the other constructor, specifying true for "passable".

this(true);

}

/\*\* Marks this cell as either being the starting or the ending cell. \*\*/

public void setEndpoint(boolean end)

{

endpoint = end;

updateAppearance();

}

/\*\*

\* Sets this cell to be passable or not passable. An input of true marks

\* the cell as passable; an input of false marks it as not passable.

\*\*/

public void setPassable(boolean pass)

{

passable = pass;

updateAppearance();

}

/\*\* Returns true if this cell is passable, or false otherwise. \*\*/

public boolean isPassable()

{

return passable;

}

/\*\* Toggles the current "passable" state of the map cell. \*\*/

public void togglePassable()

{

setPassable(!isPassable());

}

/\*\* Marks this cell as part of the path discovered by the A\* algorithm. \*\*/

public void setPath(boolean path)

{

this.path = path;

updateAppearance();

}

/\*\*

\* This helper method updates the background color to match the current

\* internal state of the cell.

\*\*/

private void updateAppearance()

{

if (passable)

{

// Passable cell. Indicate its state with a border.

setBackground(Color.WHITE);

if (endpoint)

setBackground(Color.CYAN);

else if (path)

setBackground(Color.GREEN);

}

else

{

// Impassable cell. Make it all red.

setBackground(Color.RED);

}

}

/\*\*

\* Implementation of the paint method to draw the background color into the

\* map cell.

\*\*/

protected void paintComponent(Graphics g)

{

g.setColor(getBackground());

g.fillRect(0, 0, getWidth(), getHeight());

}

}